CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 5
Date posted: 27-Feb-2026

Simulation of Process Scheduling Using Shared Memory, Signals, and Semaphores

This assignment deals with the implementation of a simulation of a priority-based round-robin scheduling algorithm. Communication among the processes involved is performed using shared-memory segments, and using signals and semaphores. Individual processes will emulate the behavior of various entities in a real-life operating system.

The scheduling algorithm

We assume that the system has a single processor. A total of n processes $P_0, P_1, P_2, \ldots, P_{n-1}$ appear to the system at various times. Each P_i carries out exactly eleven CPU bursts and ten intermediate IO bursts. The burst times are written in a text file `bursts.txt`. See the Sample-Output section for the format of this file. Each process P_i has a priority which can be one of 0 (highest priority), 1, or 2 (lowest priority).

The scheduler maintains a single FIFO queue *RQ* for all the ready processes (irrespective of their priorities). At the time of scheduling, the front of the queue is extracted, and is given the next time quantum q to run. We have $q = 10$ for processes of priority 0, $q = 5$ for processes of priority 1, and $q = 2$ for processes of priority 2 (all times are in ms). If the process currently running on the CPU finishes its burst within time q, it voluntarily releases the CPU, and goes to its next IO burst (or quits). Otherwise, the process is interrupted by a timer (a special process, see below), and the scheduler is invoked (by the interrupted process) as an ISR, in order to schedule the next process from *RQ*.

The scheduler is called in the following situations.

- A new process arrives. The new process is sent to the back of the ready queue *RQ*. If the CPU is free at that instant, the process at the front of *RQ* is dispatched for time q.
- A running process finishes its last CPU burst, and exits. If *RQ* is non-empty, then the process at its front is dispatched. If *RQ* is empty, the CPU goes idle.
- A running process finishes a CPU burst (not the last one), and goes to its next IO burst. Again depending upon whether *RQ* is empty or not, the CPU goes idle or the front of *RQ* is dispatched.
- A process waiting in its IO burst is ready after the completion of the IO. This case is handled similarly as the case of the arrival of a new process.
- A running process is interrupted by the timer. The process is enqueued to the back of *RQ*, and the front of *RQ* is dispatched for the next time quantum.

When all of the n processes exit, the application stops.

Simulation of time

We assume that no process is aware of the burst times. Although a process needs to know the data as given in the file `bursts.txt`, it only obeys the timings without analyzing them in any way. In particular, if the remaining CPU burst time of a running process is larger than the time quantum q, it is interrupted by a signal (involuntary context switch), whereas if the the process finishes its burst within time q, it releases the CPU itself (voluntary context switch).

The working of the processes is governed by a globally maintained time t residing in the shared memory. In this assignment, we assume that the unit of time is millisecond (ms), and all events happen only at integer-valued times. However, running the simulation at that speed is undesirable for two reasons. First, you cannot monitor the progress in real time (and debug your code). Second, many system resources (shared-memory and semaphore operations) need to be used, and operations on *RQ*, PCB, and other shared data need to be simulated. On one occasion, a new process is to be forked (this is a time-consuming step) in one unit of time. The actual running of different processes on your machine depends on the scheduler of your OS, and you do not have significant control over that.

In view of these, we simulate time too. First, we scale up each millisecond to a human-perceivable time period. Second, we have to wait for appropriate durations so that the operations of your simulating processes can finish at the times they are supposed to. A scale-up interval Δ and a waiting delay δ are used to make the simulation run in non-decreasing sequence of time (several events may happen at the same time though).

As mentioned earlier, the time t is discretely maintained as non-negative integer values (standing for milliseconds). The simulation starts as $t = 0$. All events will happen at integer-valued times t in future. A timer process maintains t as a shared integer variable. After every interval of $\Delta + \delta$, it increments t by one. Each other process maintains a private time τ. After a period of Δ, the process increments τ by one, and waits for the timer to update t. The extra delay of δ by the timer process gives the other processes time to engage in a wait before seeing the updated value of t. Once the timer process updates t, it notifies the waiting processes to synchronize their local times τ with the incremented t. The timer then waits for another period of δ so that the other processes get sufficient time to complete their work at that time. After this delay, the timer process sets a barrier for the next ms, and the simulation repeats.

Effectively, each ms is scaled up to the time $\Delta + 2\delta$. For moderate machines of today, you can take δ in the range of 5–25 ms, and Δ in the range 25 ms to 1 second. You should also make sure that $\Delta \gg \delta$ (like $\Delta \geqslant 5\delta$). If n (the number of user processes to simulate) is small (up to 10), then the choices $\delta = 5$ ms and $\Delta = 25$ ms should work. For larger values of n, double these choices ($\delta = 10$ ms and $\Delta = 50$ ms). A safer choice is $\delta = 100$ ms and $\Delta = 1$ second, but then your simulation will run too slowly (you can anyway experiment with these settings at the beginning of your development; in your submission, change them to smaller values that you found safe).

The synchronization of time is carried out by a semaphore SYNC. It is set to 1 at the beginning of each (simulated) ms. After a delay (use `usleep`) of Δ, each process waits until SYNC becomes 0 (use `sem_op` = 0). After the sleep of duration $\Delta + \delta$, the timer process makes a `P()` operation on SYNC (`sem_op` = –1). Subsequently, the timer waits for another interval δ (so that the waiting processes all wake up, and update τ by t). The timer then makes a `V()` operation on SYNC (`sem_op` = +1). This completes the simulation of 1 ms of time.

The processes

◆ The manager

This process will be the first to run, and the last to exit. It creates the system resources (the shared-memory segments, and the semaphores) at the beginning, initializes these resources, writes its PID in a shared-memory segment, and waits until it receives SIGINT from the process launcher (or control-C from the user). Upon the reception of this signal, the manager process removes all the resources (the shared-memory segments, and the semaphores), and exits. Write a source file `manager.c`(pp), and compile it to the executable file `manager`.

◆ The timer

The working of the timer in synchronizing processes to a common global time t is already explained above. The timer process has another role to play. At the end of the time quantum q of the currently running process, it checks whether that process is still running. If so, it sends SIGUSR1 to that process. Notice that the timer process simulates the real-life clock (a piece of hardware), and is never responsible in any other work (like scheduling). An interrupted process calls a scheduler function (as if that function is the ISR for the interrupted process). Write `timer.c`(pp), and compile it to `timer`.

◆ User processes

These processes emulate the behavior of the individual user processes $P_0, P_1, P_2, \ldots, P_{n-1}$. Compile a third source code `process.c`(pp) to an executable file `process`. Each invocation carries out the work what P_i would do. It is supplied with several command-line parameters: the serial number (i in the range 0 to $n-1$), the time of arrival (the initial value of the local time τ of P_i), the priority of P_i (0, 1, or 2), and the 21 burst times (CPU and IO) for P_i. The user process simulates the bursts by sleeping ms by ms (use `usleep` on the scaled-up value of 1 ms). After each (simulated) ms, P_i synchronizes its local time τ by the global time t, carries out the work (if any) that is to be done at that time, and then goes to sleep again for the next simulated ms.

At appropriate times, P_i invokes the scheduler function `schedule_next()` (already explained in the discussion on the scheduling algorithm). Note that user processes are solely responsible for carrying out the scheduling task on behalf of the OS. No OS process (demon) is normally involved in this task. Upon arrival or after a (timer) interrupt or after an IO-burst completion, a process enqueues itself to the ready queue RQ. If no process is currently running, the front of the queue is set to run. Note that the front may be the process itself. After a CPU-burst completion, a process does not enqueue itself to RQ, but sets the next ready process at the front of RQ to run for the next time quantum (of that process) provided that RQ is not empty at this point (otherwise the CPU goes idle). Moreover, if the CPU burst of the last running process happens to be the last one, the process exits.

The process also maintains its state in the PCB (an array indexed by the serial numbers of the user processes). For this simulation, use only the states READY, RUNNING, IO, and EXITED with the usual meanings. In the RUNNING state, the process wakes up after a (simulated) ms, and checks whether the CPU burst is over. If so, or if it is interrupted by SIGUSR1 from timer, it relinquishes control of the CPU, and goes to the IO, EXITED, or READY state as is appropriate. Otherwise, it goes to sleep for the next ms. In the IO state, a process plays no role in the scheduling effort. After each (simulated) ms, it decrements the IO-burst time by 1, and if more time is remaining, goes to sleep again. If the IO burst is complete, its state changes to READY. In reality, the running process receives an interrupt from the IO device, and performs the necessary actions on behalf of the process whose IO burst has completed. For the sake of simplicity, let this task be done by the IO-completing process for itself. That is, you do not have to simulate the IO device (like timer, a piece of hardware again) and IO interrupts.

◆ The process launcher

We need another process `launcher` compiled from `launcher.c(pp)`. This automates the creation of the user processes by reading the information stored in the input file `bursts.txt`. The arrival time of each process P_i is specified in this file. Assume that the first process arrives at $t = 0$. Subsequently, the processes are listed in the input file chronologically with respect to their arrival times. Assume also that no two processes arrive at the same time. In any case, the launcher needs to fork processes in a timely manner, so it too needs to synchronize its local time τ with the global time t maintained by the timer. This is performed in the same way as user processes do. The delay δ should be sufficient for a new user process to be born and start at the precise simulated arrival time.

At the beginning, the launcher reads the PID's of the manager and the timer processes from shared memory. The launcher then reads the input file `bursts.txt` line by line. From each line, it records the launch time, priority, and the burst durations. It then waits until the arrival time, and forks a child process. The child process `exec`'s `process` with appropriate command-line parameters (serial number, priority, and the burst times).

After the creation of all the user processes, the launcher waits for all these processes to terminate. When that happens, it sends `SIGINT` to the timer and to the manager, and itself exits. The launcher and its child processes do all the terminal printing. If (during debugging) your implementation is found not to be working as intended, you may stop it by hitting control-c. The effects of this should be the same as a normal termination (after all child processes exit), that is, the launcher should catch the signal `SIGINT`.

IPC resources

The manager creates and removes the IPC resources to be used by the other processes. The roles and the structures of these resources are now explained.

◆ Shared-memory segments

The ready queue *RQ*: This stores the FIFO queue used for scheduling. Each element in the queue will be the serial number i (an integer) of a ready process P_i. Implement a circular queue of a predefined size S based on the maximum number of processes (like 100 or 1000) that you plan to handle. If there are n user processes, the size of the queue will never exceed n. In order to distinguish between the conditions for full queue and for empty queue, use an array of size $S+1$ integers. Moreover, manipulations of RQ require the front and the back indices (integers in the range $0 \ldots S$). These may be stored at $RQ[S+1]$ and $RQ[S+2]$. So you need a shared-memory segment capable of storing $S+3$ `int` variables.

The process-control-block (PCB) table: For the purpose of this simulation, each user process P_i needs to store the following information in its PCB entry: (1) its serial number i (redundant if the table is indexed by i), (2) the PID of the instance of `process` running for P_i (the timer needs this for sending SIGUSR1 after a time quantum q of that process expires), (3) the priority of the process (needed by the scheduling function `schedule_next` for setting the time for the next timer interrupt), and (4) the state of the process (this is mostly handled by the process itself, but during an involuntary context switch, the departing process changes the state of the dispatched process from READY to RUNNING). Each entry in the PCB can be a structure of four `int` variables, and the shared-memory segment needed to store the PCB's of all the user processes should have a size sufficient to accommodate the maximum number of processes that you plan to handle.

Timer-related information *T*: This segment would store five `int` variables: (1) the global time t used by the user processes and the launcher for synchronization of their local times τ, (2) the serial number of the process current running on the CPU (or –1 if the CPU is idle), (3) the time for sending the next timer interrupt (or –1 if the CPU is idle), (4) the PID of `manager`, and (5) the PID of `timer`.

You may go for three different shared-memory segments storing *RQ*, *PCB*, and *T* separately. Alternatively, you can opt for using a single combined shared-memory segment, in which *RQ*, *PCB*, and *T* start at appropriate offsets.

- **Semaphores**

You need four semaphores in this simulation. These can be created together as a single set of four semaphores, or as four sets of single semaphores (your design choice). These semaphores are used as follows.

- One semaphore for the mutual exclusion of *RQ* (initialize to 1)
- One semaphore for the mutual exclusion of *PCB* (initialize to 1)
- One semaphore for the mutual exclusion of *T* (initialize to 1)
- The synchronization semaphore SYNC: This is used in two ways. Initialized to 0, it forces the timer to wait (and set $t = 0$) until the first process (serial number 0) performs a `V()` operation to it. Subsequently, SYNC is used for the processes to synchronize all the processes to the global time t. As explained earlier, the semaphore has the value 1 during the initial delay $\Delta + \delta$ of the timer. Meanwhile (after a delay of Δ), the launcher and the active processes start waiting until the value of SYNC becomes 0. After the delay, the timer does `P()` on this semaphore changing its value to 0, thereby waking up the waiting processes. The timer waits for time δ so that the waiting processes can proceed, and again does `V()` on SYNC for the next ms.

Signal handling

The manager, the timer, and the launcher should all catch SIGINT. The signal handler would leave all the IPC resources, and exit. The launcher would additionally send SIGINT to the manager and the timer before leaving the IPC resources. The signal handler of manager would finally remove the IPC resources before exiting. Each user process should catch SIGUSR1 to handle timer interrupts.

How to run your application

Open a terminal, and run `./manager`. The IPC resources are created and initialized by the manager. After this, the manager waits until it receives SIGINT. At the end, the manager removes the IPC resources, and exits.

Open a second terminal, and run `./timer`. The timer waits on SYNC until the first process signals this semaphore to kickstart the simulation. This process keeps on incrementing t until it receives SIGINT (from `launcher` or the user).

Open a third terminal, and run `./launcher`. All terminal printing would happen in this terminal. When all the user processes exit or the user hits control-c, the launcher sends SIGINT to the timer and the manager (so you see the shell prompt returning to the first two windows), and exits itself.

If you want to run the entire application in a single terminal, use the following commands. Here, the manager and the timer are run in the background so that the shell prompt returns for running the launcher.

```
$ ./manager &
$ ./timer &
$ ./launcher
```

Submit a single archive (zip/tar/tgz) containing the three source files (and anything else you need to pack).

Sample Output

Consider the following input file `bursts.txt` on only four processes P_0, P_1, P_2, P_3.

```
0      2  7 30  5 28  5 28 16 25 20 30  3 25  5 22 11 21 14 24  6 22 19
16     0 15 30 10 23 14 24  7 30 19 26 12 22 18 27 15 24 20 21 10 20 17
38     1 18 27 20 26  6 27  4 20  6 27 12 28  3 29 15 26 18 30 17 22 20
55     2  7 27  6 28  3 22  8 28 14 22  2 23 17 26 13 29 14 28 10 29  3
-1
```

A sample run on this file is given below. Notice that your run on the same sample file is likely to produce different outputs (from the output below, and from run to run). This is normal, because you do not have control over the scheduler running on your machine. That means that if multiple events happen at the same time *t*, sequencing those events is not under your control. Make sure that your run produces one desired output in the chronological sequence.

```
$ make run
./manager &                     // Run manager in the background
sleep 0.5s                      // Wait for half a second so that the manager can write its PID to shared memory T
./timer &                       // Run timer in the background
sleep 0.5s                      // Wait for half a second so that the timer can write its PID to shared memory T
./launcher                      // Finally run the process launcher
[0] Launcher Ready
[0] Process 0: Arrival with priority = 2
[0] Process 0: Going from READY to RUNNING with next interrupt time = 2
[2] Process 0: Interrupted
[2] Process 0: Context switch to process 0 with next interrupt time = 4
[4] Process 0: Interrupted
[4] Process 0: Context switch to process 0 with next interrupt time = 6
[6] Process 0: Interrupted
[6] Process 0: Context switch to process 0 with next interrupt time = 8
[7] Process 0: CPU burst 0 complete
[7] CPU goes idle
[16] Process 1: Arrival with priority = 0
[16] Process 1: Going from READY to RUNNING with next interrupt time = 26
[26] Process 1: Interrupted
[26] Process 1: Context switch to process 1 with next interrupt time = 36
[31] Process 1: CPU burst 0 complete
[31] CPU goes idle
[37] Process 0: IO burst 0 complete
[37] Process 0: Going from READY to RUNNING with next interrupt time = 39
[38] Process 2: Arrival with priority = 1
[39] Process 0: Interrupted
[39] Process 0: Context switch to process 2 with next interrupt time = 44
[44] Process 2: Interrupted
[44] Process 2: Context switch to process 0 with next interrupt time = 46
[46] Process 0: Interrupted
[46] Process 0: Context switch to process 2 with next interrupt time = 51
[51] Process 2: Interrupted
[51] Process 2: Context switch to process 0 with next interrupt time = 53
[52] Process 0: CPU burst 1 complete
[52] Process 2: Going from READY to RUNNING with next interrupt time = 57
[55] Process 3: Arrival with priority = 2
[57] Process 2: Interrupted
[57] Process 2: Context switch to process 3 with next interrupt time = 59
[59] Process 3: Interrupted
[59] Process 3: Context switch to process 2 with next interrupt time = 64
[61] Process 1: IO burst 0 complete
[62] Process 2: CPU burst 0 complete
[62] Process 3: Going from READY to RUNNING with next interrupt time = 64
[64] Process 3: Interrupted
[64] Process 3: Context switch to process 1 with next interrupt time = 74
[74] Process 1: CPU burst 1 complete
[74] Process 3: Going from READY to RUNNING with next interrupt time = 76
[76] Process 3: Interrupted
[76] Process 3: Context switch to process 3 with next interrupt time = 78
[77] Process 3: CPU burst 0 complete
[77] CPU goes idle
[80] Process 0: IO burst 1 complete
[80] Process 0: Going from READY to RUNNING with next interrupt time = 82
[82] Process 0: Interrupted
[82] Process 0: Context switch to process 0 with next interrupt time = 84
[84] Process 0: Interrupted
[84] Process 0: Context switch to process 0 with next interrupt time = 86
[85] Process 0: CPU burst 2 complete
[85] CPU goes idle
[89] Process 2: IO burst 0 complete
[89] Process 2: Going from READY to RUNNING with next interrupt time = 94
[94] Process 2: Interrupted
[94] Process 2: Context switch to process 2 with next interrupt time = 99
[97] Process 1: IO burst 1 complete
[99] Process 2: Interrupted
[99] Process 2: Context switch to process 1 with next interrupt time = 109
[104] Process 3: IO burst 0 complete
[109] Process 1: Interrupted
[109] Process 1: Context switch to process 2 with next interrupt time = 114
[113] Process 0: IO burst 2 complete
[114] Process 2: Interrupted
[114] Process 2: Context switch to process 3 with next interrupt time = 116
[116] Process 3: Interrupted
```

```
[116] Process 3: Context switch to process 1 with next interrupt time = 126
[120] Process 1: CPU burst 2 complete
[120] Process 0: Going from READY to RUNNING with next interrupt time = 122
[122] Process 0: Interrupted
[122] Process 0: Context switch to process 2 with next interrupt time = 127
[127] Process 2: CPU burst 1 complete
[127] Process 3: Going from READY to RUNNING with next interrupt time = 129
[129] Process 3: Interrupted
[129] Process 3: Context switch to process 0 with next interrupt time = 131
[131] Process 0: Interrupted
[131] Process 0: Context switch to process 3 with next interrupt time = 133
[133] Process 3: CPU burst 1 complete
[133] Process 0: Going from READY to RUNNING with next interrupt time = 135
[135] Process 0: Interrupted
[135] Process 0: Context switch to process 0 with next interrupt time = 137
[137] Process 0: Interrupted
[137] Process 0: Context switch to process 0 with next interrupt time = 139
[139] Process 0: Interrupted
[139] Process 0: Context switch to process 0 with next interrupt time = 141
[141] Process 0: Interrupted
[141] Process 0: Context switch to process 0 with next interrupt time = 143
[143] Process 0: Interrupted
[143] Process 0: Context switch to process 0 with next interrupt time = 145
[144] Process 1: IO burst 2 complete
[145] Process 0: CPU burst 3 complete
[145] Process 1: Going from READY to RUNNING with next interrupt time = 155
[152] Process 1: CPU burst 3 complete
[152] CPU goes idle
[153] Process 2: IO burst 1 complete
[153] Process 2: Going from READY to RUNNING with next interrupt time = 158
[158] Process 2: Interrupted
[158] Process 2: Context switch to process 2 with next interrupt time = 163
[159] Process 2: CPU burst 2 complete
[159] CPU goes idle
[161] Process 3: IO burst 1 complete
[161] Process 3: Going from READY to RUNNING with next interrupt time = 163
[163] Process 3: Interrupted
[163] Process 3: Context switch to process 3 with next interrupt time = 165
[164] Process 3: CPU burst 2 complete
[164] CPU goes idle
[170] Process 0: IO burst 3 complete
[170] Process 0: Going from READY to RUNNING with next interrupt time = 172
[172] Process 0: Interrupted
[172] Process 0: Context switch to process 0 with next interrupt time = 174
[174] Process 0: Interrupted
[174] Process 0: Context switch to process 0 with next interrupt time = 176
[176] Process 0: Interrupted
[176] Process 0: Context switch to process 0 with next interrupt time = 178
[178] Process 0: Interrupted
[178] Process 0: Context switch to process 0 with next interrupt time = 180
[180] Process 0: Interrupted
[180] Process 0: Context switch to process 0 with next interrupt time = 182
[182] Process 1: IO burst 3 complete
[182] Process 0: Interrupted
[182] Process 0: Context switch to process 1 with next interrupt time = 192
[186] Process 2: IO burst 2 complete
[186] Process 3: IO burst 2 complete
[192] Process 1: Interrupted
[192] Process 1: Context switch to process 0 with next interrupt time = 194
[194] Process 0: Interrupted
[194] Process 0: Context switch to process 2 with next interrupt time = 199
[198] Process 2: CPU burst 3 complete
[198] Process 3: Going from READY to RUNNING with next interrupt time = 200
[200] Process 3: Interrupted
[200] Process 3: Context switch to process 1 with next interrupt time = 210
[209] Process 1: CPU burst 4 complete
[209] Process 0: Going from READY to RUNNING with next interrupt time = 211
[211] Process 0: Interrupted
[211] Process 0: Context switch to process 3 with next interrupt time = 213
[213] Process 3: Interrupted
[213] Process 3: Context switch to process 0 with next interrupt time = 215
[215] Process 0: Interrupted
[215] Process 0: Context switch to process 3 with next interrupt time = 217
[217] Process 3: Interrupted
[217] Process 3: Context switch to process 0 with next interrupt time = 219
[218] Process 2: IO burst 3 complete
[219] Process 0: CPU burst 4 complete
[219] Process 3: Going from READY to RUNNING with next interrupt time = 221
[221] Process 3: CPU burst 3 complete
[221] Process 2: Going from READY to RUNNING with next interrupt time = 226
[226] Process 2: Interrupted
[226] Process 2: Context switch to process 2 with next interrupt time = 231
[227] Process 2: CPU burst 4 complete
[227] CPU goes idle
[235] Process 1: IO burst 4 complete
[235] Process 1: Going from READY to RUNNING with next interrupt time = 245
[245] Process 1: Interrupted
[245] Process 1: Context switch to process 1 with next interrupt time = 255
[247] Process 1: CPU burst 5 complete
[247] CPU goes idle
[249] Process 3: IO burst 3 complete
[249] Process 0: IO burst 4 complete
[249] Process 3: Going from READY to RUNNING with next interrupt time = 251
[251] Process 3: Interrupted
[251] Process 3: Context switch to process 0 with next interrupt time = 253
```

```
[253] Process 0: Interrupted
[253] Process 0: Context switch to process 3 with next interrupt time = 255
[254] Process 2: IO burst 4 complete
[255] Process 3: Interrupted
[255] Process 3: Context switch to process 0 with next interrupt time = 257
[256] Process 0: CPU burst 5 complete
[256] Process 2: Going from READY to RUNNING with next interrupt time = 261
[261] Process 2: Interrupted
[261] Process 2: Context switch to process 3 with next interrupt time = 263
[263] Process 3: Interrupted
[263] Process 3: Context switch to process 2 with next interrupt time = 268
[268] Process 2: Interrupted
[268] Process 2: Context switch to process 3 with next interrupt time = 270
[269] Process 1: IO burst 5 complete
[270] Process 3: Interrupted
[270] Process 3: Context switch to process 2 with next interrupt time = 275
[272] Process 2: CPU burst 5 complete
[272] Process 1: Going from READY to RUNNING with next interrupt time = 282
[281] Process 0: IO burst 5 complete
[282] Process 1: Interrupted
[282] Process 1: Context switch to process 3 with next interrupt time = 284
[284] Process 3: Interrupted
[284] Process 3: Context switch to process 0 with next interrupt time = 286
[286] Process 0: Interrupted
[286] Process 0: Context switch to process 1 with next interrupt time = 296
[294] Process 1: CPU burst 6 complete
[294] Process 3: Going from READY to RUNNING with next interrupt time = 296
[296] Process 3: Interrupted
[296] Process 3: Context switch to process 0 with next interrupt time = 298
[298] Process 0: Interrupted
[298] Process 0: Context switch to process 3 with next interrupt time = 300
[300] Process 2: IO burst 5 complete
[300] Process 3: CPU burst 4 complete
[300] Process 0: Going from READY to RUNNING with next interrupt time = 302
[301] Process 0: CPU burst 6 complete
[301] Process 2: Going from READY to RUNNING with next interrupt time = 306
[304] Process 2: CPU burst 6 complete
[304] CPU goes idle
[321] Process 1: IO burst 6 complete
[321] Process 1: Going from READY to RUNNING with next interrupt time = 331
[322] Process 3: IO burst 4 complete
[323] Process 0: IO burst 6 complete
[331] Process 1: Interrupted
[331] Process 1: Context switch to process 3 with next interrupt time = 333
[333] Process 2: IO burst 6 complete
[333] Process 3: CPU burst 5 complete
[333] Process 0: Going from READY to RUNNING with next interrupt time = 335
[335] Process 0: Interrupted
[335] Process 0: Context switch to process 1 with next interrupt time = 345
[340] Process 1: CPU burst 7 complete
[340] Process 2: Going from READY to RUNNING with next interrupt time = 345
[345] Process 2: Interrupted
[345] Process 2: Context switch to process 0 with next interrupt time = 347
[347] Process 0: Interrupted
[347] Process 0: Context switch to process 2 with next interrupt time = 352
[352] Process 2: Interrupted
[352] Process 2: Context switch to process 0 with next interrupt time = 354
[354] Process 0: Interrupted
[354] Process 0: Context switch to process 2 with next interrupt time = 359
[356] Process 3: IO burst 5 complete
[359] Process 2: CPU burst 7 complete
[359] Process 0: Going from READY to RUNNING with next interrupt time = 361
[361] Process 0: Interrupted
[361] Process 0: Context switch to process 3 with next interrupt time = 363
[363] Process 3: Interrupted
[363] Process 3: Context switch to process 0 with next interrupt time = 365
[364] Process 1: IO burst 7 complete
[365] Process 0: Interrupted
[365] Process 0: Context switch to process 3 with next interrupt time = 367
[367] Process 3: Interrupted
[367] Process 3: Context switch to process 1 with next interrupt time = 377
[377] Process 1: Interrupted
[377] Process 1: Context switch to process 0 with next interrupt time = 379
[378] Process 0: CPU burst 7 complete
[378] Process 3: Going from READY to RUNNING with next interrupt time = 380
[380] Process 3: Interrupted
[380] Process 3: Context switch to process 1 with next interrupt time = 390
[385] Process 2: IO burst 7 complete
[390] Process 1: CPU burst 8 complete
[390] Process 3: Going from READY to RUNNING with next interrupt time = 392
[392] Process 3: Interrupted
[392] Process 3: Context switch to process 2 with next interrupt time = 397
[397] Process 2: Interrupted
[397] Process 2: Context switch to process 3 with next interrupt time = 399
[399] Process 0: IO burst 7 complete
[399] Process 3: Interrupted
[399] Process 3: Context switch to process 2 with next interrupt time = 404
[404] Process 2: Interrupted
[404] Process 2: Context switch to process 0 with next interrupt time = 406
[406] Process 0: Interrupted
[406] Process 0: Context switch to process 3 with next interrupt time = 408
[408] Process 3: Interrupted
[408] Process 3: Context switch to process 2 with next interrupt time = 413
[411] Process 1: IO burst 8 complete
[413] Process 2: Interrupted
```

```
[413] Process 2: Context switch to process 0 with next interrupt time = 415
[415] Process 0: Interrupted
[415] Process 0: Context switch to process 3 with next interrupt time = 417
[417] Process 3: Interrupted
[417] Process 3: Context switch to process 1 with next interrupt time = 427
[427] Process 1: CPU burst 9 complete
[427] Process 2: Going from READY to RUNNING with next interrupt time = 432
[430] Process 2: CPU burst 8 complete
[430] Process 0: Going from READY to RUNNING with next interrupt time = 432
[432] Process 0: Interrupted
[432] Process 0: Context switch to process 3 with next interrupt time = 434
[434] Process 3: Interrupted
[434] Process 3: Context switch to process 0 with next interrupt time = 436
[436] Process 0: Interrupted
[436] Process 0: Context switch to process 3 with next interrupt time = 438
[437] Process 3: CPU burst 6 complete
[437] Process 0: Going from READY to RUNNING with next interrupt time = 439
[439] Process 0: Interrupted
[439] Process 0: Context switch to process 0 with next interrupt time = 441
[441] Process 0: Interrupted
[441] Process 0: Context switch to process 0 with next interrupt time = 443
[443] Process 0: CPU burst 8 complete
[443] CPU goes idle
[447] Process 1: IO burst 9 complete
[447] Process 1: Going from READY to RUNNING with next interrupt time = 457
[457] Process 1: Interrupted
[457] Process 1: Context switch to process 1 with next interrupt time = 467
[460] Process 2: IO burst 8 complete
[463] Process 3: IO burst 6 complete
[464] Process 1: CPU burst 10 complete
[464] Process 2: Going from READY to RUNNING with next interrupt time = 469
              [464] Process 1: Exiting
[467] Process 0: IO burst 8 complete
[469] Process 2: Interrupted
[469] Process 2: Context switch to process 3 with next interrupt time = 471
[471] Process 3: Interrupted
[471] Process 3: Context switch to process 0 with next interrupt time = 473
[473] Process 0: Interrupted
[473] Process 0: Context switch to process 2 with next interrupt time = 478
[478] Process 2: Interrupted
[478] Process 2: Context switch to process 3 with next interrupt time = 480
[480] Process 3: Interrupted
[480] Process 3: Context switch to process 0 with next interrupt time = 482
[482] Process 0: Interrupted
[482] Process 0: Context switch to process 2 with next interrupt time = 487
[487] Process 2: Interrupted
[487] Process 2: Context switch to process 3 with next interrupt time = 489
[489] Process 3: Interrupted
[489] Process 3: Context switch to process 0 with next interrupt time = 491
[491] Process 0: CPU burst 9 complete
[491] Process 2: Going from READY to RUNNING with next interrupt time = 496
[493] Process 2: CPU burst 9 complete
[493] Process 3: Going from READY to RUNNING with next interrupt time = 495
[495] Process 3: Interrupted
[495] Process 3: Context switch to process 3 with next interrupt time = 497
[497] Process 3: Interrupted
[497] Process 3: Context switch to process 3 with next interrupt time = 499
[499] Process 3: Interrupted
[499] Process 3: Context switch to process 3 with next interrupt time = 501
[500] Process 3: CPU burst 7 complete
[500] CPU goes idle
[513] Process 0: IO burst 9 complete
[513] Process 0: Going from READY to RUNNING with next interrupt time = 515
[515] Process 2: IO burst 9 complete
[515] Process 0: Interrupted
[515] Process 0: Context switch to process 2 with next interrupt time = 520
[520] Process 2: Interrupted
[520] Process 2: Context switch to process 0 with next interrupt time = 522
[522] Process 0: Interrupted
[522] Process 0: Context switch to process 2 with next interrupt time = 527
[527] Process 2: Interrupted
[527] Process 2: Context switch to process 0 with next interrupt time = 529
[529] Process 3: IO burst 7 complete
[529] Process 0: Interrupted
[529] Process 0: Context switch to process 2 with next interrupt time = 534
[534] Process 2: Interrupted
[534] Process 2: Context switch to process 3 with next interrupt time = 536
[536] Process 3: Interrupted
[536] Process 3: Context switch to process 0 with next interrupt time = 538
[538] Process 0: Interrupted
[538] Process 0: Context switch to process 2 with next interrupt time = 543
[543] Process 2: CPU burst 10 complete
[543] Process 3: Going from READY to RUNNING with next interrupt time = 545
              [543] Process 2: Exiting
[545] Process 3: Interrupted
[545] Process 3: Context switch to process 0 with next interrupt time = 547
[547] Process 0: Interrupted
[547] Process 0: Context switch to process 3 with next interrupt time = 549
[549] Process 3: Interrupted
[549] Process 3: Context switch to process 0 with next interrupt time = 551
[551] Process 0: Interrupted
[551] Process 0: Context switch to process 3 with next interrupt time = 553
[553] Process 3: Interrupted
[553] Process 3: Context switch to process 0 with next interrupt time = 555
[555] Process 0: Interrupted
```

```
[555] Process 0: Context switch to process 3 with next interrupt time = 557
[557] Process 3: Interrupted
[557] Process 3: Context switch to process 0 with next interrupt time = 559
[559] Process 0: Interrupted
[559] Process 0: Context switch to process 3 with next interrupt time = 561
[561] Process 3: Interrupted
[561] Process 3: Context switch to process 0 with next interrupt time = 563
[563] Process 0: Interrupted
[563] Process 0: Context switch to process 3 with next interrupt time = 565
[565] Process 3: CPU burst 8 complete
[565] Process 0: Going from READY to RUNNING with next interrupt time = 567
[566] Process 0: CPU burst 10 complete
[566] CPU goes idle
              [566] Process 0: Exiting
[593] Process 3: IO burst 8 complete
[593] Process 3: Going from READY to RUNNING with next interrupt time = 595
[595] Process 3: Interrupted
[595] Process 3: Context switch to process 3 with next interrupt time = 597
[597] Process 3: Interrupted
[597] Process 3: Context switch to process 3 with next interrupt time = 599
[599] Process 3: Interrupted
[599] Process 3: Context switch to process 3 with next interrupt time = 601
[601] Process 3: Interrupted
[601] Process 3: Context switch to process 3 with next interrupt time = 603
[603] Process 3: CPU burst 9 complete
[603] CPU goes idle
[632] Process 3: IO burst 9 complete
[632] Process 3: Going from READY to RUNNING with next interrupt time = 634
[634] Process 3: Interrupted
[634] Process 3: Context switch to process 3 with next interrupt time = 636
[635] Process 3: CPU burst 10 complete
[635] CPU goes idle
              [635] Process 3: Exiting
              [Launcher] All processes exited
$
```