CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 2
Date posted: 16-Jan-2026

Inter-process Communication Using Signals

It is 10am. Tatkal ticket booking for AC Chair Cars of Foonagar–Bargarh Technology Express begins. Customers $C_1, C_2, \ldots, C_n$ together log on to the booking server. The booking agent A maintains a FIFO queue Q of the waiting customers. Each customer will try to buy at least one ticket, but can buy no more than two tickets. The number of passengers for each ticket will be in the range 1– 4. Initially, t tickets are available. The agent A serves the customer requests according to the ordering imposed by the queue Q. The booking session ends when either no tickets are left, or when all of the n customers have left the reservation system (so the queue Q is empty).

This assignment deals with the development of a multi-process application to simulate the behaviors of the agent and of the individual customers. Write a file *agent.c*(*pp*) to mimic the behavior of the booking agent A, and another file *customer.c*(*pp*) to simulate the behavior of each customer. The agent program A starts first. It reads t (the initial number of tickets) and n (the number of customers) from its command-line arguments, and stores a random permutation of the customer IDs (1, 2, . . . , n) to initiate the queue Q. A then forks n child processes $C_1, C_2, \ldots, C_n$ for the n customers. Each child process execs the customer program with its ID (not PID) as the command-line parameter, and starts waiting until a signal comes from A. A waits for a while (like one second) until all the child processes are ready to participate. Subsequently, A keeps on serving booking requests as follows.

> Let C be the customer at the front of the queue Q. (C is waiting for receiving a signal from A.)
> The agent A removes C from (the front of) Q, sends `SIGUSR1` to C, and waits for a response from C.
> If C has already bought two tickets or has bought only one ticket and is not willing to buy the second ticket:
> C sends `SIGUSR2` to A, and terminates after printing a leaving message.
> Otherwise:
> C writes its ID (in the range 1 – n) and the requested number r (randomly chosen in the range 1 – 4) of tickets in a text file *request.txt*, sends `SIGUSR1` to A, and waits to know from A the status of the booking.
> If A receives `SIGUSR2` from C, it waits for the termination of C.
> If A receives `SIGUSR1` from C, it reads *request.txt* to know the requested number r of tickets.
> Let t denote the number of available tickets at that point of time.
> If $r \leqslant t$, then A decrements t by r, and sends `SIGUSR1` to C, otherwise A sends `SIGUSR2` to C.
> In both the cases, A enqueues C at the back of Q.
> C is woken up by the signal, and prints whether booking was successful or not (depending on the signal).
> C waits until it eventually comes to the front of Q again.

Write appropriate signal handlers in both the agent and the customer programs. The above interaction is implemented inside these handlers. Use the system call `pause()` to wait until a signal arrives. Avoid pausing in a signal handler, because you should not rely on the behavior of nested signal handling. Loop outside (like in `main()`) as follows.

```
while (1) pause();
```

The booking session ends in one of the following two ways.

- All customer requests are served, and all of the n customer processes have terminated. In this case, the queue Q becomes empty, and the agent A terminates too.
- The number t of available tickets reduces to 0, so no further booking is possible. In this case, A sends `SIGKILL` to all the customer processes still waiting in Q. A waits for the terminations of these processes, and then itself terminates.

Submit a single zip/tar/tgz archive containing the two source programs (and optionally a makefile).

Sample Output

In the sample output below, the agent program prints from the beginning of the lines. Each customer program, on the other hand, does an indented printing.

Do not hard-code the values of t and n. These are to be passed as command-line parameters to the agent program. Each process needs a source of randomness, and should seed the random-number generator by its own PID.

```
$ gcc -Wall -o agent agent.c
$ gcc -Wall -o customer customer.c
```

```
$ ./agent 15 5
                        Customer   1 joins the booking system
                        Customer   2 joins the booking system
                        Customer   3 joins the booking system
                        Customer   4 joins the booking system
                        Customer   5 joins the booking system

Agent: Queue = ( 4 1 5 3 2 ) Available = 15
                        Customer   4: Request for 4 tickets
                        Customer   4: Booking 1 successful

Agent: Queue = ( 1 5 3 2 4 ) Available = 11
                        Customer   1: Request for 4 tickets
                        Customer   1: Booking 1 successful

Agent: Queue = ( 5 3 2 4 1 ) Available = 7
                        Customer   5: Request for 3 tickets
                        Customer   5: Booking 1 successful

Agent: Queue = ( 3 2 4 1 5 ) Available = 4
                        Customer   3: Request for 2 tickets
                        Customer   3: Booking 1 successful

Agent: Queue = ( 2 4 1 5 3 ) Available = 2
                        Customer   2: Request for 3 tickets
                        Customer   2: Booking 1 failed

Agent: Queue = ( 4 1 5 3 2 ) Available = 2
                        Customer   4: Request for 4 tickets
                        Customer   4: Booking 2 failed

Agent: Queue = ( 1 5 3 2 4 ) Available = 2
                        Customer   1 leaves the booking system

Agent: Queue = ( 5 3 2 4 ) Available = 2
                        Customer   5: Request for 4 tickets
                        Customer   5: Booking 2 failed

Agent: Queue = ( 3 2 4 5 ) Available = 2
                        Customer   3: Request for 1 tickets
                        Customer   3: Booking 2 successful

Agent: Queue = ( 2 4 5 3 ) Available = 1
                        Customer   2: Request for 4 tickets
                        Customer   2: Booking 1 failed

Agent: Queue = ( 4 5 3 2 ) Available = 1
                        Customer   4 leaves the booking system

Agent: Queue = ( 5 3 2 ) Available = 1
                        Customer   5: Request for 1 tickets
                        Customer   5: Booking 2 successful

Agent terminates customers 3 2 5

Agent: Booking session over (no more tickets available)
```

```
$ ./agent 20 5
                        Customer   1 joins the booking system
                        Customer   2 joins the booking system
                        Customer   3 joins the booking system
                        Customer   4 joins the booking system
                        Customer   5 joins the booking system

Agent: Queue = ( 3 5 4 2 1 ) Available = 20
                        Customer   3: Request for 3 tickets
                        Customer   3: Booking 1 successful

Agent: Queue = ( 5 4 2 1 3 ) Available = 17
                        Customer   5: Request for 3 tickets
                        Customer   5: Booking 1 successful

Agent: Queue = ( 4 2 1 3 5 ) Available = 14
                        Customer   4: Request for 4 tickets
                        Customer   4: Booking 1 successful

Agent: Queue = ( 2 1 3 5 4 ) Available = 10
                        Customer   2: Request for 4 tickets
                        Customer   2: Booking 1 successful

Agent: Queue = ( 1 3 5 4 2 ) Available = 6
                        Customer   1: Request for 3 tickets
                        Customer   1: Booking 1 successful

Agent: Queue = ( 3 5 4 2 1 ) Available = 3
                        Customer   3: Request for 4 tickets
                        Customer   3: Booking 2 failed

Agent: Queue = ( 5 4 2 1 3 ) Available = 3
                        Customer   5: Request for 4 tickets
                        Customer   5: Booking 2 failed

Agent: Queue = ( 4 2 1 3 5 ) Available = 3
                        Customer   4 leaves the booking system

Agent: Queue = ( 2 1 3 5 ) Available = 3
                        Customer   2 leaves the booking system

Agent: Queue = ( 1 3 5 ) Available = 3
                        Customer   1: Request for 2 tickets
                        Customer   1: Booking 2 successful

Agent: Queue = ( 3 5 1 ) Available = 1
                        Customer   3: Request for 4 tickets
                        Customer   3: Booking 2 failed

Agent: Queue = ( 5 1 3 ) Available = 1
                        Customer   5 leaves the booking system

Agent: Queue = ( 1 3 ) Available = 1
                        Customer   1 leaves the booking system

Agent: Queue = ( 3 ) Available = 1
                        Customer   3 leaves the booking system

Agent: Booking session over (no more customers available)
```