CS39002 Operating Systems Laboratory
Spring 2024

Lab Assignment: 2
Date: 17–Jan –2024

IPC using signals

There is a simple-minded (stupid if you will) program written in *job.c*. This keeps on printing a character and sleeps for a second. If the character is supplied as an argument, that character is printed. Without any command-line argument, a random sequence of upper-case letters is printed. This is similar in spirit to the Linux command *yes*, except that *job.c* terminates after printing ten characters (so it runs for about ten seconds). The program does not handle any signals by custom-made handlers, nor does it send any signal to any process. Indeed, the code for *job.c* will be supplied to you. Use it as it is. You are not allowed to alter this code in any manner. Compile it to an executable file *job*.

Your task is to write a smart manager program *mgr.c*. The manager talks directly with the user, and based upon user input, performs one of the following tasks in each iteration of a loop.

User input	Task done by the manager
p	The manager keeps a small process table PT with 11 entries. PT[0] is reserved for the manager itself (SELF). The remaining ten entries are meant for storing information about the *job*s it initiates (see the user input **r** below). Each such entry stores the pid of that *job*, the process group id of the *job*, the current status of the *job*, and the argument supplied to the *job* for printing. The status of a *job* will be one of the following. `FINISHED` *job* has finished after printing ten characters. `TERMINATED` *job* was terminated by ^C from the user (terminal). `SUSPENDED` *job* was suspended by ^Z from the user (terminal). `KILLED` *job* was killed by the manager in the suspended state. See the sample output given at the end.
r	Start a new *job* with a randomly chosen upper-case letter for printing. Each *job* is *exec*-ed by a new child process, and runs in the foreground. As mentioned above, each *job* runs for about ten seconds. The user may press ^C or ^Z when the *job* is running. This would terminate or suspend the running *job*. Recall that the process table PT[] has 11 entries, and PT[0] is reserved for the manager. If the user asks to initiate the eleventh *job*, the manager should quit with a non-zero status. Of course, the PT entries for FINISHED, TERMINATED, and KILLED *job*s are reusable, but you do not have to do that.
c	Do nothing if there are no suspended *job*s. Otherwise ask the user about which of the currently suspended *job*s is to be resumed (continued). Upon a valid input (an index in PT[] storing the information about a *job* with SUSPENDED status), the *job* resumes running in the foreground until it either completes (a total of) ten printings and exits normally, or encounters a ^C or ^Z again from the user terminating or re-suspending the *job*.
k	Do nothing if there are no suspended *job*s. Otherwise ask the user about which of the currently suspended *job*s is to be killed. Upon a valid input (an index in PT[]), the process is prematurely killed by the manager (the remaining characters are not printed by the *job*).
h	Print a help message (see the Sample Output).
q	Quit with exit status 0. (What will happen to the would-be orphaned suspended *job*s? The manager would kill them before exiting.)

Compile the manager program to an executable named *mgr*.

In order to write the manager program, you need to take care of quite a few issues explained in detail now.

1. Running *mgr* from a shell (like bash) gives it a new process-group id (typically the same as the pid of the *mgr* process). Any child process (*job*) *fork*-ed by *mgr* would by default have the same process group id as *mgr* (even after *exec*). If the user presses ^C or ^Z, the corresponding signal (SIGINT or SIGTSTP) goes to all the processes in the process group. That is, a user input ^C will terminate not only the child *job*, but the parent *mgr* too. But we want ^C or ^Z to affect only the child *job*, not the parent process (*mgr*). So the parent process should write its own signal handlers to deal with these signals. Note that *job.c* does not use any signal handler.

2. The manager *mgr* can kill (user command **k**) or resume (user command **c**) a suspended *job* by sending SIGKILL or SIGCONT to it from the respective signal handlers of *mgr*.

3. Suppose that a parent process (*mgr* in our case) has some custom-made signal handlers defined before forking a child process (a *job* in our case). If the child process *exec*s to run a new program, the signal handlers of the parent are not available to the child process. This is natural, because the functions private to the parent program are not accessible to the child program. That is, even if *mgr* has its own handlers for ^C and ^Z, child *job*s are not affected by those, and are terminated or killed as desired.

4. Apparently, your problem is solved. Well, almost! There is a subtle catch. Suppose that *mgr* runs a *job*, and while *job* is still running, the user presses ^Z. With the solution presented so far, this will suspend the *job* but not *mgr*. Later, upon user's request (**r**), another *job* starts running—call it *job′*. Before *job′* finishes, the user presses ^C causing *job′* to terminate. Note that *mgr* handles this ^C by its own handler. But what happens to the supended *job*. Because it is suspended, ^C does not immediately affect *job*. Later, if the user plans to resume *job* (using the command **c**), it wakes up, receives the pending ^C, and terminates without finishing its remaining work. This should not happen. In other words, no suspended process should be affected by ^C from the user.

5. A way to solve this problem is outline now. This is what is typically done by a shell like bash, and that is why your running *mgr* gets its pgid as its pid (not the pgid or pid of the shell). Before *exec*ing *job*, a child process changes its process-group id by making the system call *setpgid*(). A safe new process-group id for the child is its own pid (obtained by *getpid*()). Since ^C and ^Z apply only to *mgr*'s process-group id, a suspended *job* will no longer receive these signals.

6. But then, a running *job* too will not be affected by ^C or ^Z (because these keyboard inputs apply only to the processes with the *mgr*'s pgid). That too is undesirable. This problem too has a solution. The parent has its custom-made ^C and ^Z handlers. Moreover, it knows which *job* is currently running under it. So *mgr*'s signal-handler routines can be designed to send the approprate signals (SIGINT or SIGTSTP) to the currently running *job*. Unfotunately, signal handlers do not accept any custom-made parameters, so you have to use global variables. Too bad! Isn't it?

That's all! Go ahead, and incorporate the above suggestions one by one in the sequence given. See what happens before incorporating each suggestion and after doing it. Stop when your program can supply an output as given in the sample below.

Submit your final *mgr.c*.

Sample output

```
$ gcc -Wall -o job job.c
$ gcc -Wall -o mgr mgr.c
$ ./mgr
mgr> h
   Command : Action
      c    : Continue a suspended job
      h    : Print this help message
      k    : Kill a suspended job
      p    : Print the process table
      q    : Quit
      r    : Run a new job
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
mgr> r
Running job M
M M M M M M M M M M
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
mgr> r
Running job Y
Y Y ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       SUSPENDED       job Y
mgr> r
Running job H
H H H H H H ^C
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       SUSPENDED       job Y
3     10932       10932       TERMINATED      job H
mgr> r
Running job E
E E E E E ^C
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       SUSPENDED       job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
mgr> r
Running job P
P P ^Z
mgr> r
Running job A
A A A A ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       SUSPENDED       job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       SUSPENDED       job P
6     10942       10942       SUSPENDED       job A
mgr> ^C
mgr> ^Z
mgr> c
Suspended jobs: 2, 5, 6 (Pick one): 5
P P P P ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       SUSPENDED       job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       SUSPENDED       job P
6     10942       10942       SUSPENDED       job A
mgr> k
Suspended jobs: 2, 5, 6 (Pick one): 2
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       SUSPENDED       job P
6     10942       10942       SUSPENDED       job A
```

```
mgr> r
Running job H
H H H ^C
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       SUSPENDED       job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
mgr> r
Running job W
W W ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       SUSPENDED       job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
8     11003       11003       SUSPENDED       job W
mgr> c
Suspended jobs: 5, 6, 8 (Pick one): 5
P P P P
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       FINISHED        job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
8     11003       11003       SUSPENDED       job W
mgr> c
Suspended jobs: 6, 8 (Pick one): 8
W W ^C
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       FINISHED        job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
8     11003       11003       TERMINATED      job W
mgr> c
Suspended jobs: 6 (Pick one): 6
A A ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       FINISHED        job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
8     11003       11003       TERMINATED      job W
mgr> r
Running job F
F F F F F ^C
mgr> r
Running job T
T T T T ^Z
mgr> p
NO    PID         PGID        STATUS          NAME
0     10906       10906       SELF            mgr
1     10909       10909       FINISHED        job M
2     10910       10910       KILLED          job Y
3     10932       10932       TERMINATED      job H
4     10939       10939       TERMINATED      job E
5     10941       10941       FINISHED        job P
6     10942       10942       SUSPENDED       job A
7     11002       11002       TERMINATED      job H
8     11003       11003       TERMINATED      job W
9     11017       11017       TERMINATED      job F
10    11018       11018       SUSPENDED       job T
mgr> r
Process table is full. Quiting...
$
```