Kanchi

CS 451 – Operating Systems
Shortest Remaining Time First Scheduler

Project Description

In this assignment, you will be writing a scheduler called `srtfScheduler` to schedule processes to run, using shortest remaining time first scheduling. Given the running time information about processes, you will write a scheduler that runs the processes in the order of remaining run time of the process. Essentially, when a process is running, if another process arrives whose run time is less than the remaining time of the currently running process, then the running process is removed from CPU and the process that newly arrived is given the CPU.

The srtfscheduler creates one child process for each process in the file, and schedules them according to the shortest remaining time scheduling algorithm. It ensures that, at every instant of time, the process that is currently running has the *least* amount of remaining run time compared to any other process in the system. A process is taken off the CPU when either when the running process finishes its burst, or, a newly arrived process has lower burst time that the remaining time of the currently running process

The input to the scheduler consists of `m` lines, each line contains a process number `(0 to m-1)`, arrival time of the process, and, the CPU burst of each process, separated by one or more spaces. There will be a header line in the input file which is not processed. The arrival times and CPU bursts are given in seconds and will be whole numbers. The processes are listed in the increasing order of their arrival times and no processes arrive in fractional number of seconds such as 2.5. Also multiple processes *may not* arrive at the same instant. There will be at most 10 entries in the file. The entries in each line may be separated by spaces or tabs and any number of them. Your program should handle this with no errors. There are no errors expected in the input file. The name of the file containing the input is read from the command line.

The scheduler keeps time to determine which process needs to be scheduled next on the CPU. When a process is scheduled to run for the first time, it is forked and it is run. When a new process arrives that needs to be scheduled, the running process is *suspended* and the new process is run. When it is time for the suspended process to run, it is *resumed* and it is run. When a process finishes its entire burst, the scheduler *terminates* that process.

Thus the scheduler switches processes on the CPU by *suspending* the one process, and *resuming* another process. The suspensions and resumptions are done by sending signals

to the corresponding child processes. The scheduler keeps time by using an interval timer, as described later in this document.

For simplicity, each child process executes the same code. It will start with a *random* 10 digit number, say 1234567890, and simply find the next prime number higher than the starting number. The primality checking algorithm we use is a trivial one, in which number *n* is decided to be prime, if it is not divisible by any number between 2 to *n*. This algorithm is chosen so that CPU is kept busy during the burst time of the process. Keeping the starting number large will keep the CPU busy. Note that you have to use `long unsigned int` as the type for the variable storing the number being checked for primality.

Each child process should print to standard out, first time when it is *started*, each time it is *suspended*, each time it is *resumed*, and when it is *completed*. When a child process first gets the CPU, it should print the first number that it is checking to verify primality. Each time a child process is suspended or resumed, it should print the largest number that it determined to be prime. When the process finally finishes its burst it should print the largest number it found to be prime. No intermediate prime numbers should be printed.

The output from your scheduler should include its scheduling activities: A line when it schedules a process for the first time, i.e forks, each time the scheduler suspends a process or resumes a process, and each time it terminates a process. Each line of printout should include the current time in seconds that is kept by the scheduler.

SAMPLE RUN:

Assume input.txt contains the following:

PID	AT	BURST
0	1	7
1	2	4
2	4	6
3	6	4
4	7	2

```
%./srtfScheuler input.txt

Scheduler: Time Now: 1 second
Scheduling to Process 0 (Pid 1314) whose remaining time is 7
seconds.
```

(When a process scheduled for the first time, it prints the following)

```
Process 0: my PID is 1314: I just got started. I am starting with
the number 233343434 to find the next prime number.

Scheduler: Time Now: 2 seconds
Suspending Process 0 with remaining time of 6 seconds and
Resuming Process 1 with remaining time of 4 seconds
```

(When a process is suspended… it prints the following)

```
Process 0: my PID is 1314: I am about to be suspended... Highest
prime number I found is 5754853343.

Process 1: my PID is 1315: I just got started. I am starting with
the number 9848288302 to find the next prime number.


Scheduler: Time Now: 4 seconds
New process 2 with remaining time of 4 seconds has arrived


Scheduler: Time Now: 6 seconds
Terminating Process 1
New process 3 with remaining time of 4 seconds has arrived
Scheduling process 3


Etc.
```

(When a process ends... it prints the following)

```
Process 0: my PID is 1314: I completed my task and I am exiting.
Highest prime number I found is 1000000000063.
```

All the data in the output above are made-up. It shows you the format and what is expected. For example, you could be printing a different prime number.

Attacking the Problem:

Prior Knowledge: Everyone should have prior experience of using fork, exec and signal handling. There is some help provided below on these topics. If you need additional help, please stop by and I will be happy to help.

Suspending and Resuming Child Processes:

Your main (parent) program will schedule the highest priority child by suspending the child process that is currently running and resuming the child process with highest

priority. This is done by sending signal of **SIGTSTP (not SIGSTP)** to the child to suspend and SIGCONT to resume. Both of these signals can be handled in the child so that the child can do the printout when the signal is received. Note that sending SIGSTP will stop the child but this signal cannot have a handler. So, child will stop but will be unable to print when it receives the signal. The parent can use the *kill*(pid, signal_num) function call to send the signal to the child by providing it pid in the kill call.

To handle signals you need to set up signal handler. You need write a function, and set up signal handler so that the function runs whenever the specified signal is received.

It can be done as follows: If you additional help, please contact me:

```
struct sigaction sa;

 /* Install timer_handler as the signal handler for SIGALRM. */
 memset (&sa, 0, sizeof (sa));
 sa.sa_handler = &timer_handler;
 sigaction (SIGALRM, &sa, NULL);
```

Keeping track of time in the scheduler:

To keep time, the parent uses the real interval timer which measures the real "wall clock time". This is the timer that will be used by the scheduler to schedule, suspend and resume processes. When the timer goes off it sends SIGALRM which can then handle the suspension and resumption of process as needed. The timer is set using code along these lines..

```
struct itimerval timer;

 /* Install timer_handler as the signal handler for SIGALRM. */

/* The timer goes off 1 second after installation of the timer.
timer.it_value.tv_sec = 1;
timer.it_value.tv_usec =0;


/* ... and every 1 second after that. */
 timer.it_interval.tv_sec = 1;
 timer.it_interval.tv_usec =0;


/* Start a reak timer. It counts down whenever this process is
   executing. */
 setitimer (ITIMER_REAL, &timer, NULL);

 while (1) { }
```

…

How to start a child process by the scheduler

Set up a real timer in the scheduler as specified above. In the signal handler, `fork` the desired processes at the appropriate time. After you fork the child, use one of the `exec` suite of calls to execute the child process in the child. Of course you should have `prime.o` file ready to execute, so you can use in inside exec. Send at least two parameters in exec. One for sending the process number and one for priority. This will help the child process to print this information. This will be in argv of the child.

How to stop a child process by the scheduler

When a child process has completed its burst (as calculated by the scheduler), send a signal SIGTERM. The child prints an output (shown above) and exits the system.

How to write code for child processes.

In a separate file, say prime.c, you should write a main function as if it is an independent program.

It should print its first message saying that it has started (shown above), then stay in an infinite loop trying to find large prime number. When it receives signals it does the printout.

The child should stay in a forever loop finding higher and higher primes and handling arriving signals (SIGTSTP, SIGCONT and SIGTERM).

Notes:

1. Since there are multiple processes running on the CPU, and do not have a dedicated CPU, when the scheduler (main) processes schedules a child, the child may not run immediately. However, for the purposes of this assignment, we will assume that child process starts immediately after it is scheduled by the scheduler. So, when the parent process counts *k* real seconds, we assume that the child that is currently scheduled ran for *k* real seconds.

2. Global variable usage is permitted. You will need them. In OS class, global variable usage is common. Enjoy using them!

3. Always end every printf with "\n". It completes the buffer and makes printout happen. Otherwise signals will forbid a printout from completion.

4. As you develop you will find issues where there are too many processes 'hanging around' the system. You may use *ps –aef* to see which processes are running and kill undesired processes during debugging.

5. The child processes get killed in the end (alas!) so the scheduler need not issue waitpid.

Original Work

This assignment must be the original work of you. Unless you have explicit permission, you may not include code from any other source or have anyone else write code for you. You are welcome use the code I have given you in this document, however.

Use of unattributed code is considered plagiarism and will result in academic misconduct proceedings as described in the Syllabus.