Simulated Operating Systems
Phase 2

Admin Command Processing via Fork, Pipe, and Signal

We simulate the administrator interface and activities in admin.c. Specific admin commands can be found in admin.c and are also listed in the following table.

Action	Parameters	System actions
T	-	Terminate the entire system
s (submit)	fnum	Submit a new process, should be shifted to client program
x (execute)	-	Execute a program from ready queue till some event stops it
y (execute)	r	Repeat what is done for x r times
r (register)	-	Dump registers
q (queue)	-	Dump ready queue and endIO list
p (PCB)	-	Dump PCB for every process in the system
m (memory)	-	Dump the page table of each process, for all processes
f (memory frame)	-	Dump the metadata of all memory frames, frame by frame
n (memory)	-	Dump the contents of the entire memory, frame by frame
e (timer events)	-	Dump timer event list
t (term)	-	Dump the terminal request queue
w (swap)	-	Dump the swap request queue

In current admin.c, the input command is read in by scanf. This means if the admin issues “y 100000”, thenit is not possible to issue any observation command while the system is in execution for 100000 rounds. We would like to change the admin code so that it runs asynchronously with the execution of the processes so that during process execution, admin can issue commands to observe the system behaviors.

The goal above shall be achieved by writing a new admin interface process (**adminUI.c** which compiles to **admin.exe**). Upon initialization, adminUI.c **fork**s a child process which runs simOS (better not to use exec). After fork, the child simOS process should initialize the system, run execute_process for a large number of rounds, and call system_exit to exit simOS. To simulate real computer systems, we would like to let simOS execute processes in an infinite loop. However, if your program is not designed properly, you may end up having a process that never terminates and you need to kill it externally. So, we give a command line input toadmin.exe, **numR**, which is the number of rounds execute_process should be activated (instead of having an infinite loop).

When the adminUI.c reads a command from the administrator, it should send the command to simOS. We use pipe to pass the command from admin interface to simOS. But this way, simOS still need to be blocked on aread from pipe. We use Unix system interrupt (signal and signal handler) to achieve the goal. Before admin forks, two pipes shall be created to support two-way communication between the admin interface process and the simOS process. The admin commands will be issued with a signal (not simOS interrupt) and a pipe to simOSand processed by a Unix signal handler (not simOS interrupt handler) in simOS. The flow of the admin.exe program is: (1) read admin command from administrator, (2) sends a signal to simOS process via **kill**(…) system call, (3) sends the command to simOS process via the parent write pipe, (4) waits for the response from simOS via the parent read pipe, and (5) display the simOS output to the monitor.

The interface for admin.exe should be the same as the original simOS.exe, except that the administrator canno longer issue “x” or “y” commands.

You need to make a few changes to simOS to enable the communication with adminUI.c and to allow admin commands being executed while the processes are in execution.

(A) In simOS, you need to define a new interrupt type and (say adcmdInterrupt) and give it a proper value in simos.h. The purpose of this interrupt is to let the system switch to serve the admin command when it is set. Then, you need to modify handle_interrupt in cpu.c to handle adcmdInterrupt. The interrupt handler simply needs to activate the current admin command processing function. Thus, you should implement the handler in admin.c.

(B) In admin.c (in simOS), you should remove the admin commands “x” and “y”. You do not need to call execute_process for a single round. The function execute_process_iteratively should now be invoked at the system initialization time. Also, you need to capture a Unix system signal (not simOS interrupt) by associating a

signal handler to a selected signal number (use the **signal** system call). You can choose either of the signalnumbers: SIGINT, SIGRTMIN, or SIGRTMAX for the purpose. This can also be incorporated in admin.c.

Upon receiving a signal, your signal handler should: (1) read the admin command from the child read pipe, (2) raise adcmdInterrupt.

(C) The output from simOS should now be directed to a pipe and sent back to the admin process for displaying. The admin process can simply create a pipe for the response communication and use the pipe as thefile descriptor for all the dump functions. The admin process can then receive the responses and print them outon the monitor.