Scheduling

1.1 Implementation of a circular return timer in the user space

A round-robin timer is required. The timer is executed as a parent process, in the user space, dividing the computing time into processes-children. To control the processes the timer will use the SIGSTOP and SIGCONT signals to stop and activate each process, respectively.

Each process is executed for a period of time equal to the quantum time tq. If the process is terminated before the end of the quantum time, the timer removes it from the queue of ready processes and activates the next one. If the time quantum expires without the process having completed its execution, then it is stopped, placed at the end of the ready process queue, and the next one is activated.

The timer function is required to be asynchronous, based on signals. A kernel space timer is activated by a timer break. Respectively, the timer in question will use the SIGALRM and SIGCHLD signals to be activated in the following two cases:

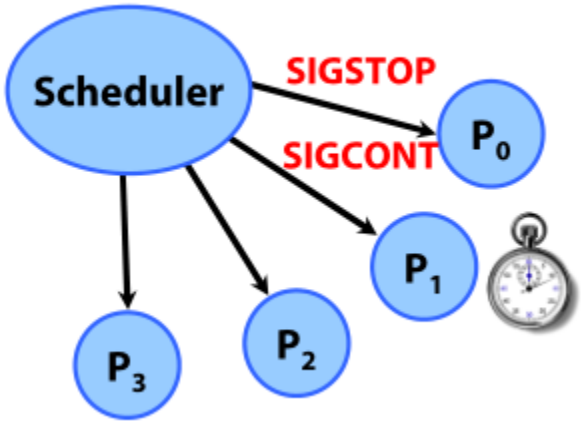

Figure 1

• Quantum time expiration: When the time quantum expires, the timer stops the current process. This case corresponds to handling the SIGALRM signal.

•Process Pause / End: When the current process dies or stops because its time quantum has expired, the timer selects the next one from the queue, sets the timer to deliver a SIGALRM signal after tq seconds, and activates it. This case corresponds to manipulation of the SIGCHLD signal.

The final delivery is the schedule of the timer, which will be executed with arguments that are executable to date:

$ ./scheduler prog prog prog prog

For each argument a process is constructed that executes the corresponding program. Each process is assigned a serial number id. Once the processes are created, the scheduling process begins. The timer displays appropriate messages when activating, stopping, and terminating processes. When all processes have completed their execution, it terminates.

In the / home / oslab / code / sched directory you are given: the program prog.c, which will run in the scheduling processes, execve - example.c for using the execve () system call and a timer skeleton, scheduler.c.

Note: A process running under strace ignores the SIGSTOP signal. Therefore, avoid the use of strace on children, since the scheduling is done with SIGSTOP and SIGCONT. Instead, you can run the timer under the strace without any problems.

1.2 Timer function control via bark

The extension of the timer of the previous query is requested, in order to support the control of its operation through a shell program. The user of the system has the ability to request dynamic creation and termination of processes, interacting with the cortical program.

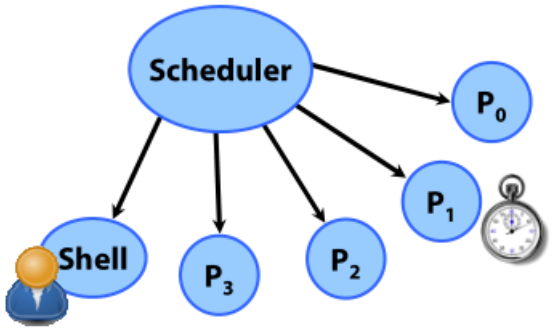

Figure 2

The cortex receives commands from the user, constructs requests of appropriate form which it sends to the time router, receives responses that inform for the outcome of their execution (success / failure) and informs the user about their result.

The shell has four commands:

• Command 'p': The router prints to the output list of under-executed processes, which shows the process id number of the process, the PID and its name. In addition, the current process is highlighted.

• Command 'k': Accepts an id of a process (attention: not PID) and asks the timer to terminate it.

• Command 'e': Arguments the name of an executable in the current directory, e.g. prog2 and requests the creation of a new process from the timer, in which this executable will run.

• Command 'q': The shell terminates.

Final deliverable is a new version of the timer, which will serve the requests of the cortex. The shell is asked to be dated along with the other processes, being in the execution queue and receiving time quantum according to the RR algorithm. The timer will shut down when all the processes it handles end, including the cortex.

You are given the shell program, shell.c and the request.h file which defines the format of requests from the shell to the scheduler based on the struct request_struct structure. To support the cortex, a shell-to-router communication mechanism and vice versa is necessary. This mechanism is given to the scheduler - shell.c file, which is the skeleton of the requested implementation.

1.3 Implement priorities in the timer

The timer extension of the previous query is requested to support two priority classes: HIGH and LOW. The scheduling algorithm changes as follows: if there are HIGH priority processes, only those are executed using round-robin. Otherwise, LOW processes are routed using cyclic reset. All processes are created with LOW priority. Changing the priority of a process will be done with the following shell commands:

• 'H' ('l') Argues the id of a process and sets its priority to HIGH (LOW).