Ho chi Minh City University Of Technology
Faculty of Computer Science & Engineering

Lab 4a Scheduling
Course: Operating Systems

Lecturer: Hoang Le Hai Thanh

October 30, 2022

Goal:

- Practice the scheduling algorithms in Operating System.

Content:

- Know how to schedule Linux tasks using Cron
- Implement a simple OS scheduling emulator

Requirement:

- Student need to review the theory of scheduling and multithreading programming techniques.
- Make sure your OS has the Cron installed and running by running the following command `crontab -l`

1 Linux Cron Jobs

1.1 Introduction to Cron

Cron is a scheduling daemon that executes tasks at specified intervals. These tasks are called cron jobs and are mostly used to automate system maintenance or administration.

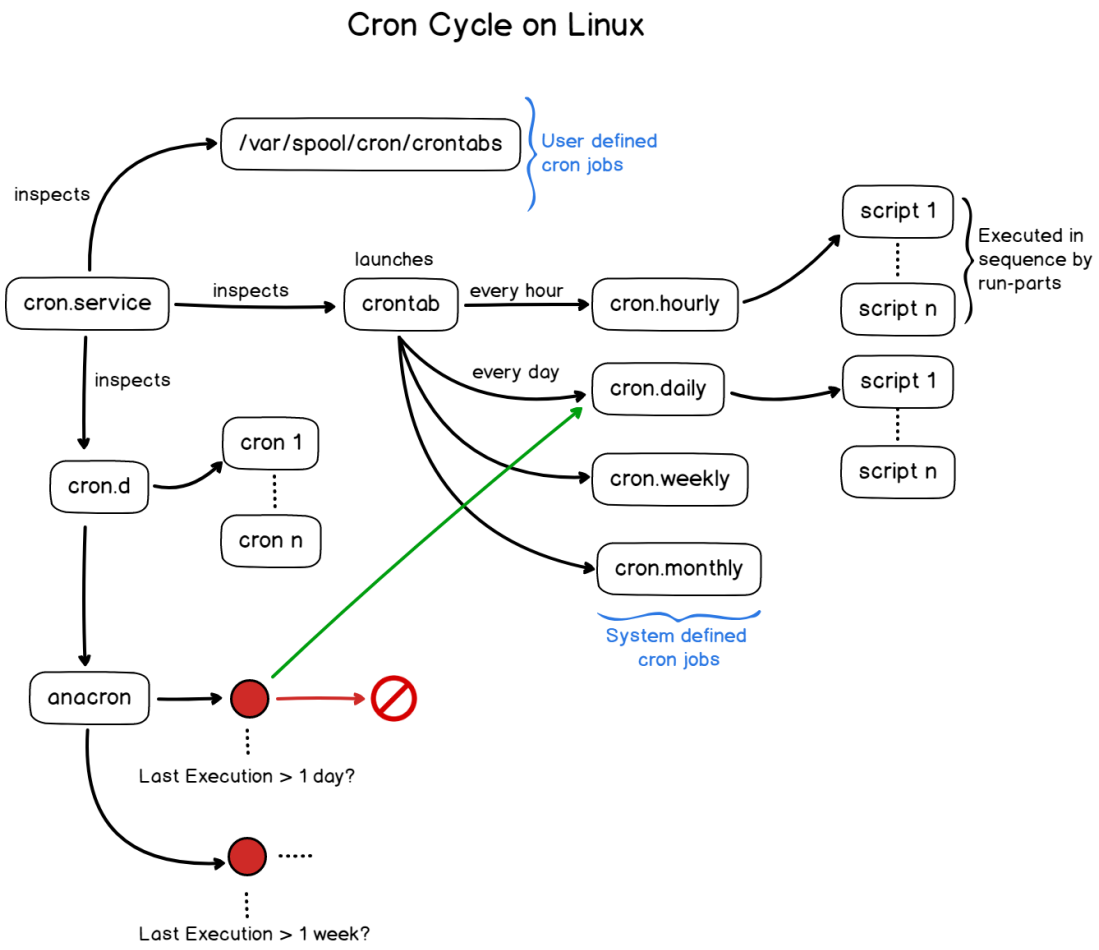

Figure 1.1: Cron Cycle

For example, we can set up a cron job to automate repetitive tasks such as backing up databases or data, updating the system with the latest security patches, checking disk space usage, sending emails, etc.

The cron jobs can be scheduled to run by a minute, hour, day of the month, month, day of the week, or any combination of these. For Ubuntu, we can inspect the cron service by running the following command:

```
$ sudo systemctl status cron.service
```

1.2 Crontab

Crontab (cron table) is a text file that specifies the schedule of cron jobs. There are two types of crontab files. The system-wide crontab files and individual user crontab files. Users' crontab files are named according to the user's name, and their location is at the `/var/spool/cron/crontabs` directory on Ubuntu. The `/etc/crontab` file and the scripts inside the `/etc/cron.d` directory are system-wide crontab files that can be edited only by the system administrators.

Fgure 1.1 describes the cron cycle. Cron will inspect the user defined cron jobs and execute them if needed. It will also inspect the crontab file where several default cron jobs are defined by default.

Those default cron jobs are scripts that instructs your host to verify every minute, every hour, every day and every week specific folders and to execute the scripts that are inside them.

Finally, the `cron.d` directory is inspected. The `cron.d` may contain custom cron files and it also contains a very important file which is the anacron cron file.

1.3 Crontab Syntax and Operators

Each line in the user crontab file contains six fields separated by a space followed by the command to be run:

```
* * * * * command(s) output
- - - - -
| | | | |
| | | | ------ Day of week (0 - 7) (Sunday=0 or 7) (Mon - Sun)
| | | -------- Month (1 - 12)
| | ---------- Day of month (1 - 31)
| ------------ Hour (0 - 23)
-------------- Minute (0 - 59)
```

The detail of a cron command are:

- The first five fields `* * * * *` specify the time/date and recurrence of the job.
- In the second section, the command specifies the location and script you want to run.

- The final segment output is optional. It defines how the system notifies the user of the job completion. Cron will issue an email if there is any output from the cron job. Cron jobs are meant to not produce any output if everything is ok.

The first five fields may contain one or more values, separated by a comma or a range of values separated by a hyphen.

- "*" - The asterisk operator means any value or always. If you have the asterisk symbol in the Hour field, it means the task will be performed each hour.
- "," - The comma operator allows you to specify a list of values for repetition. For example, if you have `1,3,5` in the Hour field, the task will run at 1 am, 3 am, and 5 am.
- "-" The hyphen operator allows you to specify a range of values. If you have `1-5` in the Day of the week field, the task will run every weekday (From Monday to Friday).
- "/" - The slash operator allows you to specify values that will be repeated over a certain interval between them. For example, if you have `*/4` in the Hour field, it means the action will be performed every four hours. It is same as specifying `0,4,8,12,16,20`. Instead of an asterisk before the slash operator, you can also use a range of values, `1-30/10` means the same as `1,11,21`.

There are several special Cron schedule macros used to specify common intervals. We can use these shortcuts in place of the five-column date specification:

- @*yearly* (or @*annually*) - Run the specified task once a year at midnight (12:00 am) of the 1st of January. Equivalent to `0 0 1 1 *`.
- @*monthly* - Run the specified task once a month at midnight on the first day of the month. Equivalent to `0 0 1 * *`.
- @*weekly* - Run the specified task once a week at midnight on Sunday. Equivalent to `0 0 * * 0`.
- @*daily* - Run the specified task once a day at midnight. Equivalent to `0 0 * * *`.
- @*hourly* - Run the specified task once an hour at the beginning of the hour. Equivalent to `0 * * * *`.
- @*reboot* - Run the specified task at the system startup (boot-time).

Example:

1. `10 10 1 * * /path/to/script.sh`: A cron job that executes every 10:10 AM each month on the first day.
2. `23 0-23/2 * * * /path/to/scripts.sh`: Execute the script.sh at 23 minutes after midnight, 2 am, 4 am..., everyday

Exercise: Convert the following intervals to crontab presentation:

- Every Monday at 08:30
- Every workday, every 30 minutes, from 8:15 to 17:45
- Last day of every month at 17:30

1.4 Linux Crontab Command

The crontab command allows you to install, view, or open a crontab file for editing:

- `crontab -e` - Edit crontab file, or create one if it doesn't already exist.
- `crontab -l` - Display crontab file contents.
- `crontab -r` - Remove your current crontab file.
- `crontab -i` - Remove your current crontab file with a prompt before removal.
- `crontab -u <username>` - Edit other user crontab file. This option requires system administrator privileges.

The cron daemon automatically sets several environment variables::

- The default path is set to `PATH=/usr/bin:/bin`. If the command you are executing is not present in the cron-specified path, you can either use the absolute path to the command or change the cron `PATH` variable. You can't implicitly append `PATH` as you would do with a regular script.
- The default shell is set to `/bin/sh`. To change to a different shell, use the `SHELL` variable.
- Cron invokes the command from the user's home directory. The `HOME` variable can be set in the crontab.

1.5 Practice with CronTab

In this lab, we will look at an example of how to schedule a simple script with a cron job. First, we will create a script called `date-script.sh` in our `HOME` folder to print the system date and time and appends it to a file. The content of our script is shown below:

```
#!/bin/sh

$ echo $(date) >> date-out.txt
```

Don't forget to make the script executable by using the `chmod` command. Then we define our job in the CronTab. To open the crontab configuration file for the current user, enter the following command:

```
$ crontab -e
```

We can add any number of scheduled tasks, one per line. In this case, we want to make our job run every minute, so we will add the following command (please change the `user` of the absolute path to your Linux username):

```
*/1 * * * * /home/user/date-script.sh
```

Wait for some minutes and verify our cron job by checking the content of our output file:

```
$ cat date-out.txt

Thu 20 Oct 2022 04:02:01 PM UTC
Thu 20 Oct 2022 04:03:01 PM UTC
Thu 20 Oct 2022 04:04:02 PM UTC
Thu 20 Oct 2022 04:05:02 PM UTC
Thu 20 Oct 2022 04:06:01 PM UTC
```

Or we can check Cron log inside the system log at `/var/log/syslog`:

```
$ cat /var/log/syslog | grep CRON

...
Oct 20 16:00:01 n1 CRON[1666129]: (CRON) info (No MTA installed
    , discarding output)
Oct 20 16:01:01 n1 CRON[1666304]: (user) CMD (date-script.sh)
Oct 20 16:01:01 n1 CRON[1666303]: (CRON) info (No MTA installed
    , discarding output)
Oct 20 16:02:01 n1 CRON[1666478]: (user) CMD (/home/user/date-
    script.sh)
Oct 20 16:03:01 n1 CRON[1666642]: (user) CMD (/home/user/date-
    script.sh)
Oct 20 16:04:01 n1 CRON[1666815]: (user) CMD (/home/user/date-
    script.sh)
```

2 LINUX PROCESS SCHEDULING

2.1 A Simple Scheduler Emulator

Next, we will dive deeper into the Scheduling concept using our previous Linux Thread coding experience. Look at the following figure 2.1, we have a simple scheduler that works as follows:

1. Scheduling policy parameters and CPU processes are described in the `input.txt` file. The first line contains the timeslot parameter for the Round Robin algorithm and the number of emulation jobs which are described in the below lines.

2. Each process has at least two required parameters: The arrival time (first column) and the burst time (second column).

3. For each new process, the loadtask() will create a new process and assign a new PCB to it. Then all processes are stored in the `in_queue`.

4. When the emulation begins, at each checking interval, the `loader()` will look into the `in_queue` to push any arrived processes to the `ready_queue` (arrival time = current time)

5. The `CPU()` runs concurrently with the `loader()` and executes any process from the head of the `ready_queue`.

6. The CPU runs processes in a Round-Robin style. Each process is allowed to run up to a given period (timeslot) before being returned to the `ready_queue` or terminated if its burst time remains 0 (this process is done).

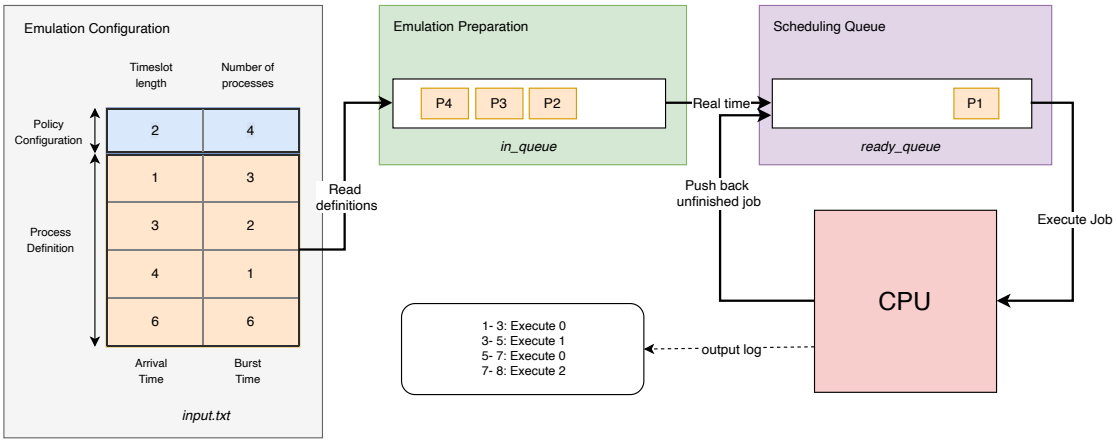

Figure 2.1: Model of emulating the scheduling algorithm in OS.

2.2 In class exercise

Although the OS is designed to work on multiple processors, in this part of this lab, we assume the system has only one processor. This lab requires you to develop a simple Linux Scheduler emulator. You will be provided a sample source code with some important incomplete methods.

You have to complete these methods which are marked by the `//TODO` directives:

1. The `en_queue` and the `de_queue` methods of the queue source file `queue.c`. Please remind your linked list knowledge and multithreading techniques.

2. The behavior of the `CPU()` when executes a process (in `sched.c`):

    a) Calculate the time that the `CPU()` (`exec_time`) will spend to execute a process which is loaded into the CPU, The `exec_time` must not exceed the `timeslot` threshold. You also have to update the process `burst_time` to avoid infinite execution loops.

    b) Check whether the process has finished its execution. If the process is done, `free` the allocated memory of the process object (`pcb_t`). Otherwise, reinsert the process to the `ready_queue`

Then create a Makefile with a `sched` target that prints the output. To know how run your code, you can examine the provided `test.sh` script.

***Note:** You need to consider the data structure which is declared in `structs.h`. This file declares the attributes and the data structure of `process` and `queue`.

3 Homework

The above Round Robin policy does not care about the process priority. In your homework, we will update our previous emulator to support preemptive algorithms such as Shorted-Job-First (SJF). To achieve this goal, you have to complete these requirements:

1. Include an additional `priority` field as the third column in the `input.txt` as described below.

```
2  4
0  7  1
2  4  2
4  1  2
5  4  1
```

2. Update the process struct definition.

3. Change the original Round-Robin algorithm to the preemptive algorithms.

Your work must also include your own implementation of the in class coding exercise requirements.

Reversion History

Revision	Date	Author	Description
1.0	2016	PD Nguyen, DH Nguyen, MT Chung	First Version
2.0	2022	Thanh Hoang Le Hai	Add Cron Job and Update OS scheduler