OPERATING SYSTEM 1

Lecture 10

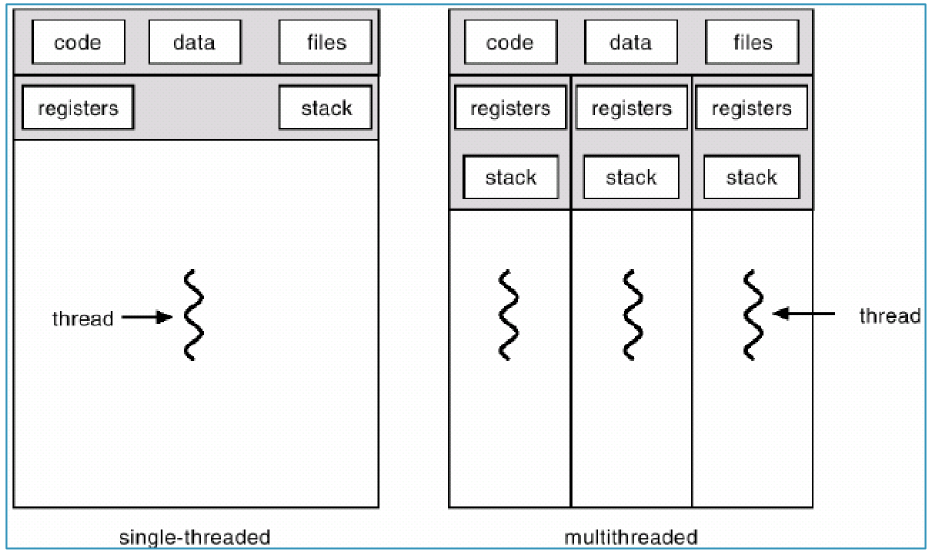
Eng. Joud Khattab

Process Management

- Definition.
- Foreground Processes.
- Background Processes.
- Listing Running Processes.
- Stopping Processes.

Definition

- A process refers to a program in execution.
- A process is a running instance of a program.
- A process is made up of the program instruction, data read from files, other programs or input from a system user.



Definition

- When you execute a program on your Unix system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.
- Whenever you issue a command in Unix, it creates, or starts, a new process. When you tried out the ls command to list the directory contents, you started a process.

Definition

- The operating system tracks processes through a five-digit ID number known as the **PID** or the **process ID**. Each process in the system has a unique **PID**.
- **PIDs** eventually repeat because all the possible numbers are used up and the next **PID** rolls or starts over. At any point of time, no two processes with the same **PID** exist in the system because it is the **PID** that Unix uses to track each process.

Definition

- When you start a process (run a command), there are two ways you can run it:
  1. Foreground Processes
  2. Background Processes

FOREGROUND PROCESSES

Foreground Processes

- By default, every process that you start runs in the foreground.
- It gets its input from the keyboard and sends its output to the screen.
- While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

BACKGROUND PROCESSES

Background Processes

- A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.
- The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!
- The simplest way to start a background process is to add an ampersand (&) at the end of the command.

Background Processes

- **Command:**
  - ./someProcessA &
  - ./someProcessB &

- **Description:**
  - This will start someProcessA and someProcessB and then exit (probably before either of the processes have finished.

Waiting Background Processes

- **Command:**
  - ./someProcessA &
  - ./someProcessB &
  - wait

- **Description:**
  - The script will wait for the background processes to finish.
    - wait can wait on a specific PID (which you could get with something like PID=$! after spawning a background process).
    - Or if you don't give it any parameters, it will wait until all background processes have finished:

Killing Background Processes

- **Command:**
  - sleep 100 &
  - ps
  - kill 1234

- **Description:**
  - Kill command can stop any background process by given its PID.

From Background to Foreground

- **Command:**
  - sleep 100 &
  - fg
  - Bg
  - Jobs

- **Description:**
  - Bg: make a foreground process to run in background
    - usage: type 'ctrl+z' and then 'bg <job id>'
  - Fg: make background process as foreground process
    - Usage: fg [jobid]
  - Jobs: displays the names and ids of background jobs
    - Usage: jobs

Listing Running Processes

- It is easy to see your own processes by running the **ps** (process status)

- **Commands:**
  - $ps
  - ID TTY TIME CMD
  - 18358 ttyp3 00:00:00 sh
  - 18361 ttyp3 00:01:31 abiword
  - 18789 ttyp3 00:00:00 ps

Listing Running Processes

Option	Description
-a	Shows information about all users
-x	Shows information about processes without terminals
-u	Shows additional information like -f option
-e	Displays extended information

Special Characters

1.	$?	is the most recent foreground pipeline exit status.
2.	$$	PID of the current shell.
3.	$#	number of arguments passed to current script.
4.	$!	is the PID of the most recent background command.

BASH SCRIPTS EXAMPLES

Sequential Runs

Program

```
#! /bin/bash
task(){
  sleep 0.5; echo "$1";
}
for thing in a b c d e f g; do
  task "$thing"
done
```

Parallel Runs

Program

```
#! /bin/bash
task(){
  sleep 0.5; echo "$1";
}
for thing in a b c d e f g; do
 task "$thing" &
done
```

Parallel Runs and Wait

Program

```
#! /bin/bash
task(){
  sleep 0.5; echo "$1";
}
for thing in a b c d e f g; do
 task "$thing" &
done
wait
```

Check Path Kind

Program

```
#! /bin/bash
echo hello &
important_pid=$!

PASSED=$1
if [ -d "$PASSED" ] ; then
   echo "$PASSED is a directory" &
else
   if [ -f "$PASSED" ]; then
      echo "$PASSED is a file" &
  fi
fi

wait $important_pid
echo Important task finished

wait
echo All tasks finished
```

HW

Search for a specific value on a 100 element array using multiple processes