Processes

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

Contents

Process

What?

Program $\neq$ Process

What?

Program $\neq$ Process

- Program is **passive**,

What?

Program $\neq$ Process

- Program is **passive**, stored on disk as an executable file

What?

Program ≠ Process

- Program is **passive**, stored on disk as an executable file
  - e.g. `/bin/ls`

```
$ ls -la /bin/ls
-rwxr-xr-x 1 root root 118280 Mar 14  2015 /bin/ls
```

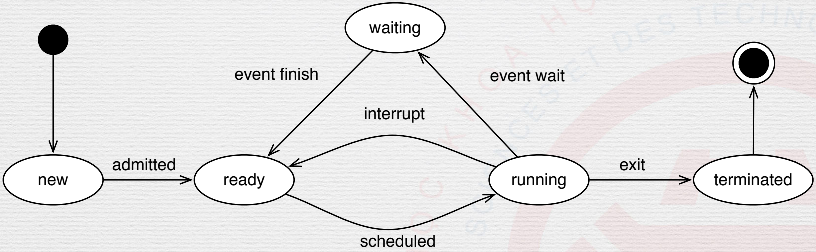
What: Process

- Process is a program in execution state

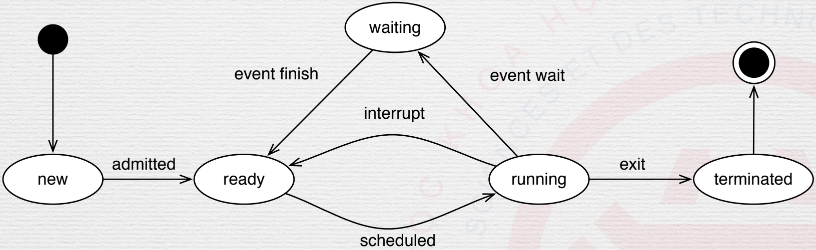
What: Process

- Process is a program in execution state (**active**)
- Created by the system or a parent process
- Uniquely identified (PID)

Process States



Process States



- **new**: process has just been created
- **ready**: waiting to be assigned (scheduled) to a processor
- **running**: it's executing instructions
- **waiting**: waiting for some events to occur
- **terminated**: finished execution

Process Control Block

- Represents a process
- Stored in memory
- Not accessible to process, only for kernel's process schedulers (later)

Process Control Block

- Represents a process
- Stored in memory
- Not accessible to process, only for kernel's process schedulers (later)
- Contains
  - Process ID
  - Process state (new/ready/running/waiting/terminated)
  - Processor state (program counter, registers)
  - File descriptors
  - Scheduling information (next section)
  - Accounting information (limits)

Process Scheduling

```
/*
 * If the new process paused because it was
 * swapped out, set the stack level to the last call
 * to savu(u_ssav).  This means that the return
 * which is executed immediately after the call to aretu
 * actually returns from the last routine which did
 * the savu.
 *
 * You are not expected to understand this.
 */
if(rp->p_flag&SSWAP) {
        rp->p_flag =& ~SSWAP;
        aretu(u.u_ssav);
}
```

Source: Unix v6 Source Code, line 2230-2243

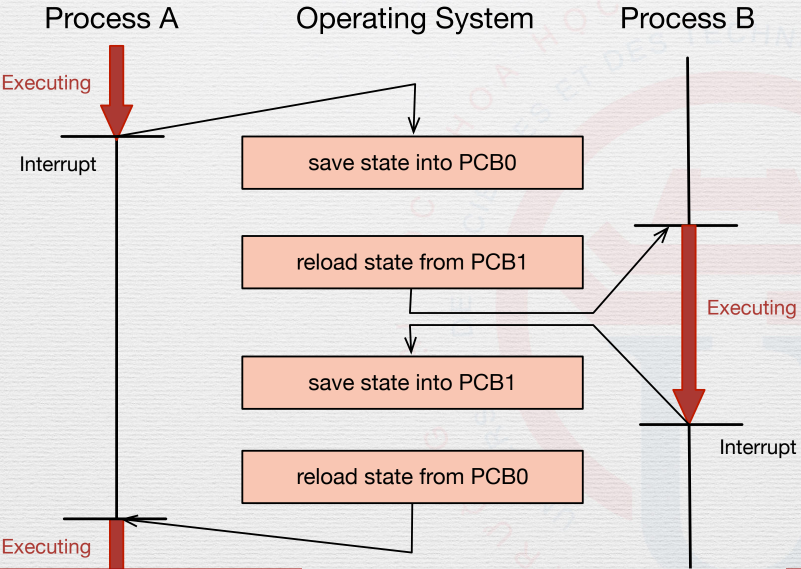
Process Scheduling

- Multiple processes running at the same time.
- Process scheduler is a part that decides which processes to be executed at a certain time.

Process Scheduling Types

- Pause running processes
  - Preemption: OS forcely pauses running processes
  - Non-preemption (also cooperation): processes willing to pause itself
- Duration between each «switch»
  - Short term scheduler: milliseconds (fast, responsive)
  - Long term scheduler: seconds/minutes (batch jobs)

Context Switch



Context Switch

- Switch between processes
    - Save data of old process
    - Load previously saved data of new process
- Context switch is overhead
    - No work done for processes during context switch
    - Time slice (time between each switch) is hardware-limited

Scheduling Algorithms

Scheduling Criteria

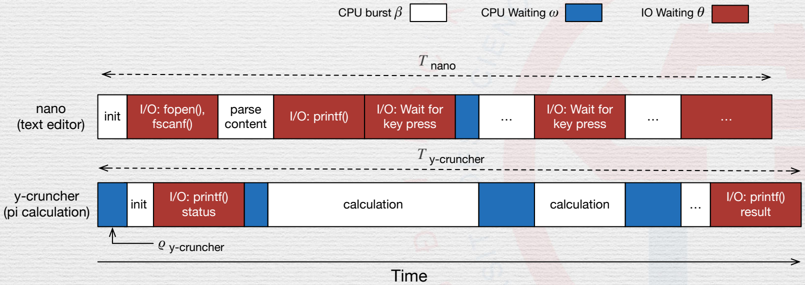
- Fair: no starvation
- Minimize waiting time
- Maximize efficiency (jobs/time)
- Prioritization

Scheduling Concepts

- Timing
  - **Burst time** β : CPU time needed for a short task
  - **CPU Waiting time** ω : time a process in the **ready** queue
  - **I/O time** θ : time a process in the **waiting** queue
  - **Turnaround time** T : total time to execute a process, from start to end, including waiting times (in **ready** and **waiting** queues)
  - **Response time** ρ: time from a submitted request until first response

Scheduling Concepts

Burst time β, CPU Waiting time ω, I/O time θ, Turnaround time T, Response time ρ.



$$T = \sum \beta_i + \sum \omega_j + \sum \theta_k$$

Scheduling Concepts

- Others
  - **CPU utilization**: percentage of CPU usage
  - **Throughput**: number of processes completing their execution per time unit
  - **Priority**: an integer number for each process, indicating its importance

Scheduling Goals

- Idle = waste of energy, so...
- Maximize
  - CPU utilization
  - Throughput
- Minimize
  - Turnaround time: for calculation
  - **Waiting time**: for typical desktop systems
  - Response time: for server systems

Algorithms

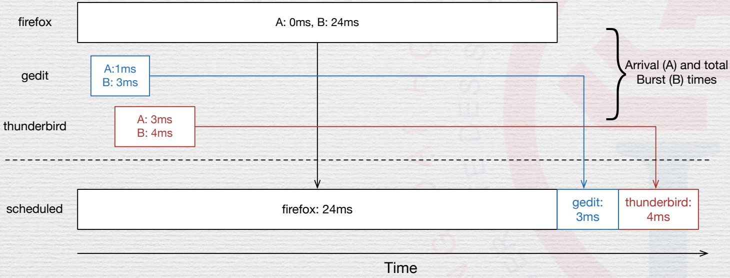
- **First Come, First Served**
- Shortest-Job-First
- Shortest-Remaining-Time-First
- Round Robin
- Multilevel Queue
- Multilevel Feedback Queue

First Come, First Served: What?

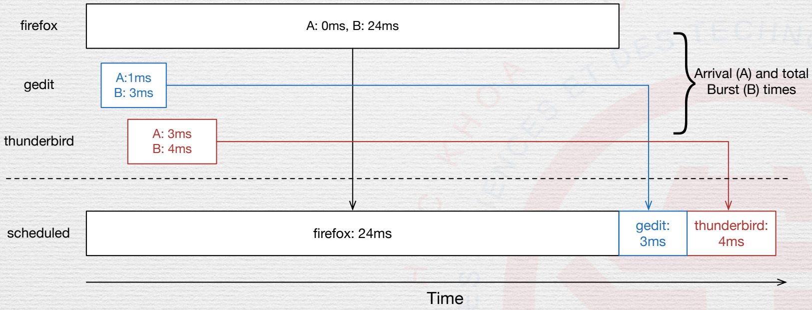
- **Non-preemptive**
- **Non-priority**
- Allocate CPU to next process in ready queue
  - **based on the process arrival time**
- Wait till it finishes CPU usage
  - I/O wait
  - Voluntarily stop
- Switch to next process

First Come, First Served: How?

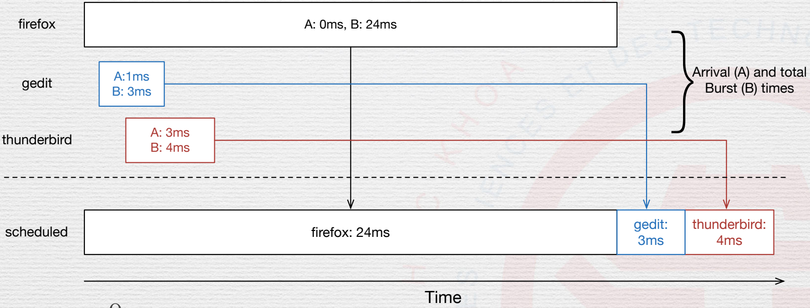
Arrival time (ms)	Process	Burst time (ms)
0	firefox	24
1	gedit	3
3	thunderbird	4



First Come, First Served: How?



First Come, First Served: How?



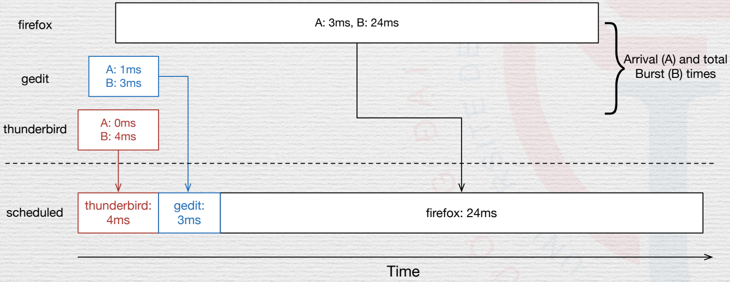
$$\omega_{firefox} = 0$$
$$\omega_{gedit} = 23$$
$$\omega_{thunderbird} = 24$$

$$\overline{\omega}_{fcfs} = \frac{0+23+24}{3} = 15.67$$

First Come, First Served: How?

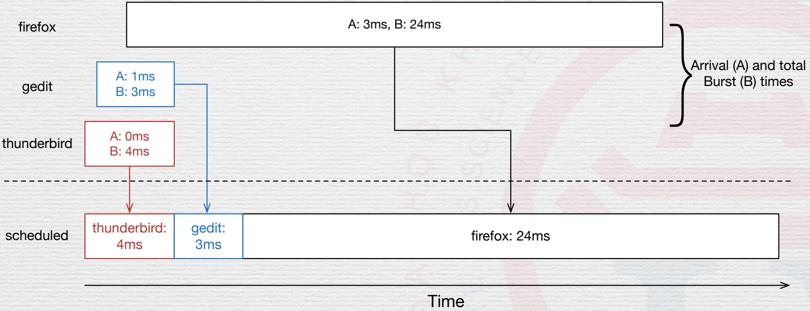
If arrival time is reversed?

Arrival time (ms)	Process	Burst time (ms)
0	thunderbird	4
1	gedit	3
3	firefox	24



First Come, First Served: How?

If arrival time is reversed?



$$\omega_{thunderbird} = 0$$
$$\omega_{gedit} = 3$$
$$\omega_{firefox} = 4$$

$$\overline{\omega}_{fcfs} = \frac{0+3+4}{3} = 2.33$$

Algorithms

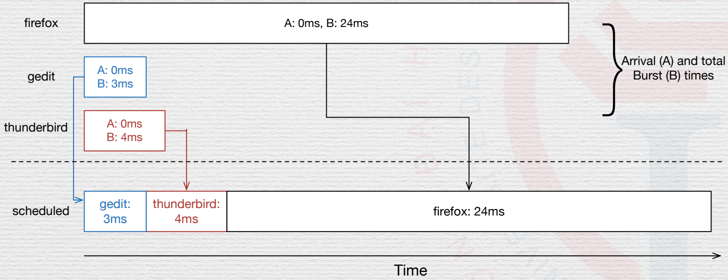
- First Come, First Served
- **Shortest-Job-First**
- Shortest-Remaining-Time-First
- Round Robin
- Multilevel Queue
- Multilevel Feedback Queue

Shortest-Job-First: What?

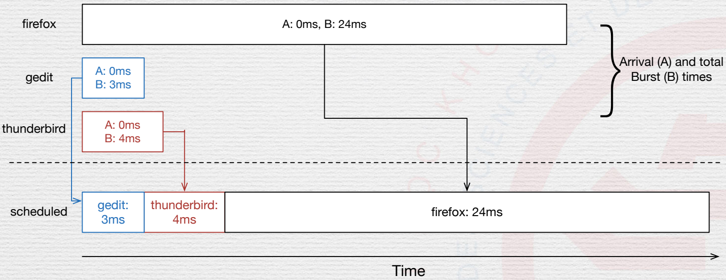
- **Non-preemptive**
- **Priority**
- Allocate CPU to next process in ready queue
  - NOT based on the process arrival time
  - based on the estimated CPU burst
  - **choose the process with shortest estimated burst**
    - If there are two processes with shortest bursts, FCFS
    - That's priority
- Optimal for waiting time $\overline{\omega}_{sjf}$
- Problem: processes with long CPU bursts will rarely be scheduled

Shortest-Job-First: How?

Arrival time (ms)	Process	Burst time (ms)
0	firefox	24
0	gedit	3
0	thunderbird	4



Shortest-Job-First: How?



$$\omega_{gedit} = 0$$

$$\omega_{thunderbird} = 3$$

$$\omega_{firefox} = 7$$

$$\overline{\omega}_{sjf} = \frac{0+3+7}{3} = 3.33$$

Algorithms

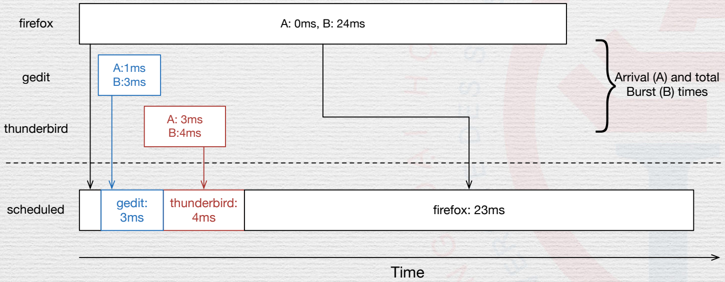
- First Come, First Served
- Shortest-Job-First
- **Shortest-Remaining-Time-First**
- Round Robin
- Multilevel Queue
- Multilevel Feedback Queue

Shortest-Remaining-Time-First: What?

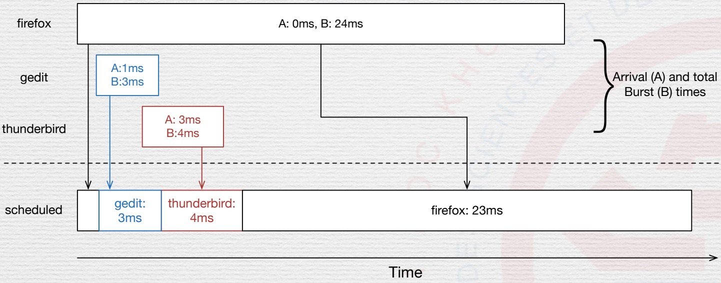
- **Preemptive**
- **Priority**
- Each preemption, allocate CPU to next process in ready queue
  - NOT based on the process arrival time
  - based on the estimate **remaining** CPU burst
  - choose the process with shortest **remaining** burst
    - If there are two processes with shortest remaining bursts, FCFS
    - That's priority
- Also called preemptive Shortest-Job-First

Shortest-Remaining-Time-First: Example

Arrival time (ms)	Process	Burst time (ms)
0	firefox	24
1	gedit	3
3	thunderbird	4



Shortest-Remaining-Time-First: Example



$$\omega_{firefox} = 0 + 7$$
$$\omega_{gedit} = 0$$
$$\omega_{thunderbird} = 1$$

$$\overline{\omega}_{srtf} = \frac{7+0+1}{3} = 2.33$$

Algorithms

- First Come, First Served
- Shortest-Job-First
- Shortest-Remaining-Time-First
- **Round Robin**
- Multilevel Queue
- Multilevel Feedback Queue

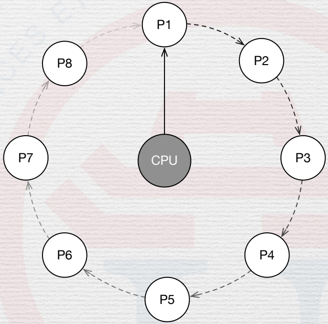
Round Robin: What?

- **Preemptive**
- **Non-priority**
- Similar to FCFS, but add periodical preemption
    - Each process has a time slice (duration)
        - Fixed, or
        - Dynamic
    - After time slice finishes
        - Process is preempted
        - Put to end of ready queue

Round Robin: What?

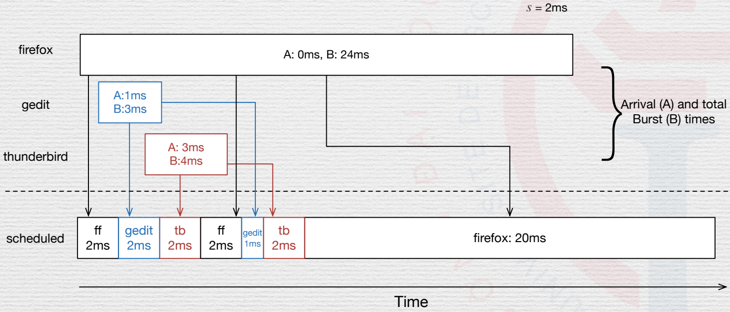
- Good for interactive systems
- Better average response time ρ
- n: number of processes in ready queue
- s: time slice (also called quantum)
- $\rightarrow$ Waiting time **per round**

$$\omega_r = (n-1) * s$$

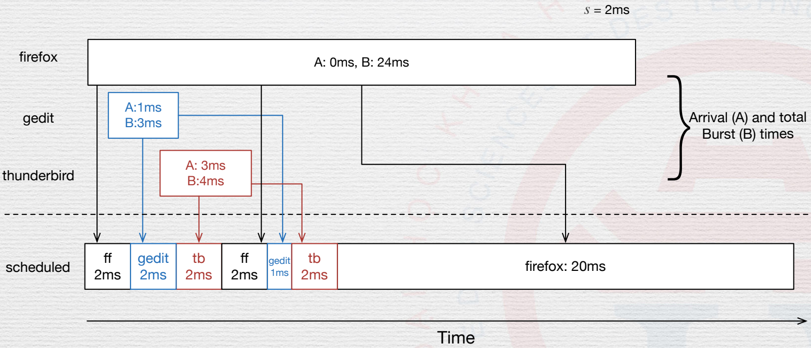


Round Robin: Example

Arrival time (ms)	Process	Burst time (ms)
0	firefox	24
1	gedit	3
3	thunderbird	4



Round Robin: Example



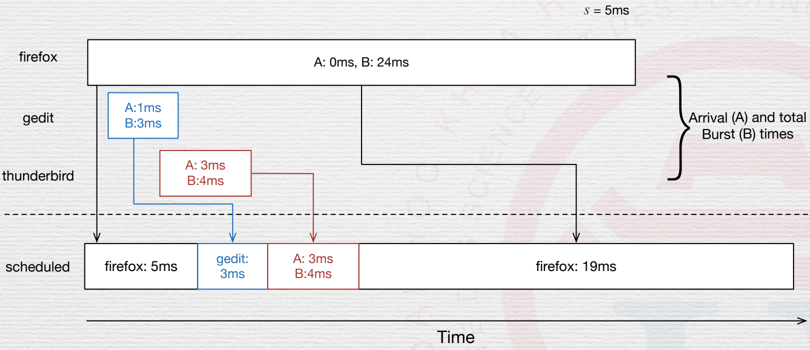
$$\omega_{firefox} = 0 + 4 + 3 = 7$$
$$\omega_{gedit} = 1 + 4 = 5$$
$$\omega_{thunderbird} = 1 + 3 = 4$$

$$\overline{\omega}_{rr} = \frac{7+5+4}{3} = 5.33$$

Round Robin: Time Slice (s)

- Smaller s, more context switches
- Too small s, too many preemption ($\sim$ context switches)
  - Remind: context switch does nothing productive
  - Overhead
- Too large s, close to FCFS

Round Robin: Time Slice (s)



- Only 3 context switches total
- 2 context switches for firefox

Algorithms

- First Come, First Served
- Shortest-Job-First
- Shortest-Remaining-Time-First
- Round Robin
- **Multilevel Queue**
- Multilevel Feedback Queue

Multilevel Queue: What?

- Combination of previous algorithms
- Ready queue is split into several subqueues
  - Based on characteristics of processes
  - **Permanently** assign them to different subqueues
  - Example: building Linux kernel while surfing web and listening to music
- Each queue uses different scheduling algorithm
  - Example: RR for foreground subqueue, FCFS for background subqueue.

Multilevel Queue: What?

- Problem: different subqueues, choose which subqueue to schedule?
  - System processes: sshd, postfix, ...
  - Interactive processes: firefox, chrome, ...
  - Background processes: gcc, make, wget, ...
- Solution
  - Subqueue priority
  - Subqueue time slice

Multilevel Queue: Subqueue Priority

- Category each subqueue with a priority
  - High priority: system processes - sshd, postfix, ...
  - Normal priority: interactive processes - firefox, chrome, ...
  - Low priority: background processes - gcc, make, wget, ...
- Finish all scheduled processes in high priority subqueue, then move to subqueue with higher priority
- Problem:
  - « Starvation »: low priority processes never get CPU

Algorithms

- First Come, First Served
- Shortest-Job-First
- Shortest-Remaining-Time-First
- Round Robin
- Multilevel Queue
- **Multilevel Feedback Queue**

Multilevel Feedback Queue: What?

- An improvement of Multilevel Queue
- Processes can be migrated from a subqueue to another
  - MLQ: processes are **permanently** assigned to subqueue
- Parameters
  - Number of subqueues
  - Scheduling algorithm for each queue
  - Rule to migrate processes

Multilevel Feedback Queue: Example

- « Completely Fair Scheduler »
- « Red-black tree »
- Linux default scheduler, since 2.6.23 (Oct 2007)
- **All** devices
- **All** platforms

Scheduling Algorithms Summary

Algorithm	Preempt?	Priority?	Note
First Come, First Served	No	No	Depends on arrival time
Shortest-Job-First	No	Yes	Low waiting time ω
Shortest-Remaining-Time-First	Yes	Yes	Preemptive SJF, low ω
Round Robin	Yes	No	Low response time ρ
Multilevel Queue	Depends	Depends	Several subqueues, permanent
Multilevel Feedback Queue	Depends	Depends	Several subqueues, migrate

Exercise 2: Scheduling

- Answer the exercise (next slide)
- Write a short report (text file, LaTeX or MarkDown is preferred, **NOT** Word document)
  - Name it « 02.report.scheduling.txt » (or .tex)
  - Write your answers
- Push your report to corresponding forked Github repository

Exercise 2: Scheduling

Process	Arrival Time (ms)	Burst Time (ms)
P_1	0.0	5
P_2	1.0	3
P_3	5.5	2
P_4	6.8	1

Exercise 2: Scheduling

1. Draw Gantt chart of these proceses with FCFS, SJF, SRTF, RR ($s = 1ms$)
   - ASCII art is preferred!
2. What is the average waiting time these processes, using...
   - FCFS
   - SJF
   - SRTF
   - RR
3. What is the average turnaround time these processes, using...
   - FCFS
   - SJF

fork-exec

Process Creation

- Start a new process == Create a new process
  - Create new child process
    - Can create child process → grand child process
  - Dependent on OS, parent and child can share
    - **All** resources: opened files, devices, etc...
    - **Some** resources: opened files only
    - **No** resource
- A fully loaded system will have a process tree

Process Creation

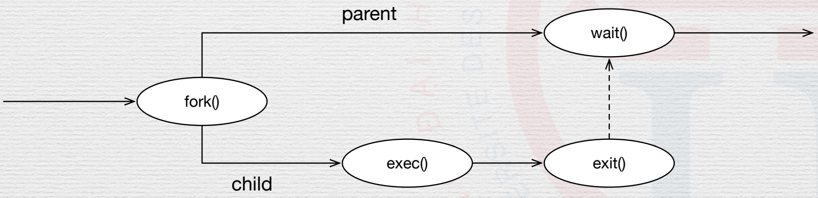
```
$ pstree -A
init-+-acpid
     |-cron
     |-daemon---mpt-statusd---sleep
     |-dbus-daemon
     |-dovecot-+-anvil
     |         |-config
     |         `-log
     |-master-+-pickup
     |        |-qmgr
     |        `-tlsmgr
     |-mysqld_safe---mysqld---23*[{mysqld}]
     |-php5-fpm---2*[php5-fpm]
     |-proftpd
     |-screen---bash---python2---{python2}
     |-sshd-+-sshd---sshd---bash---pstree
     |      `-sshd---sshd
     |-udevd---2*[udevd]
     `-znc---{znc}
```

Process Creation on UNIX/Linux

- New processes are not created from scratch
- Two steps
  - `fork()`
  - `exec()`

Process Creation on UNIX/Linux

- New processes are not created from scratch
- Two steps
  - `fork()`
  - `exec()`



Process Creation on UNIX/Linux

- `fork()`
  - Perfectly «clone» current process to a new process
    - Open files
    - Register states
    - Memory allocations
    - **Except** process id
  - Who's who? Parent? Child?
    - Use return value of `fork()`

```
pid_t fork(void);
```

Process Creation on UNIX/Linux

- Parent: `fork()` returns process id of child
- Child: `fork()` returns 0
- Example

```
#include <unistd.h>
#include <stdio.h>
int main() {
    printf("Main before fork()\n");
    int pid = fork();
    if (pid == 0) printf("I am child after fork()\n");
    else printf("I am parent after fork(), child is %d\n", pid);
    return 0;
}
```

```
$ ./dofork
Main before fork()
I am parent after fork(), child is 2378
I am child after fork()
```

Process Creation on UNIX/Linux

- `exec()`
  - Load an executable binary to replace current process image
  - A family of functions.
  - Ask `man`

```
int execl(...);
int execle(...);
int execlp(...);
int execv(...);
int execvp(const char *file, char *const argv[]);
int execvP(...);
```

Process Creation on UNIX/Linux

- exec() example

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Going to launch ps -ef\n");
    char *args[]= { "/bin/ps", "-ef" , NULL};
    execvp("/bin/ps", args);
    return 0;
}
```

```
$ ./doexec
Going to launch ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 Oct19 ?        00:01:34 init [2]
root           2     0  0 Oct19 ?        00:00:01 [kthreadd]
...
dovecot     2933  2899  0 Oct19 ?        00:00:02 dovecot/anvil
root        2934  2899  0 Oct19 ?        00:00:24 dovecot/log
root        3095     1  0 Oct19 ?        00:04:57 /usr/lib/postfix/master
```

Process Creation on UNIX/Linux

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Main before fork()\n");
    int pid = fork();
    if (pid == 0) {
        printf("I am child after fork(), launching ps -ef\n");
        char *args[]= { "/bin/ps", "-ef" , NULL};
        execvp("/bin/ps", args);
        printf("Finished launching ps -ef\n");
    }
    else printf("I am parent after fork(), child is %d\n", pid);
    return 0;
}
```

```
$ ./forkexec
Main before fork()
I am parent after fork(), child is 12278
I am child after fork(), launching ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
```

Process Creation on UNIX/Linux

- Questions
  - Why was "Finished launching ps -ef" **not** printed?
  - Why not a single call like Windows `CreateProcess()` or `WinExec()`?
  - Or, why does UNIX separate `fork()` then `exec()`?

Process Creation on UNIX/Linux

- Questions
  - Why was "Finished launching ps -ef" **not** printed?
  - Why not a single call like Windows `CreateProcess()` or `WinExec()`?
  - Or, why does UNIX separate `fork()` then `exec()`?
- Answers
  - Process memory was replaced, the command to print is not there anymore

Process Creation on UNIX/Linux

- Questions
  - Why was "Finished launching ps -ef" **not** printed?
  - Why not a single call like Windows `CreateProcess()` or `WinExec()`?
  - Or, why does UNIX separate `fork()` then `exec()`?
- Answers
  - Process memory was replaced, the command to print is not there anymore
  - `fork()` has different purposes as well, not only in process creation
    - Worker processes (apache, chrome...)

Process Creation on UNIX/Linux

- Synchronization between parent and child
  - `wait()`: wait for termination of a child
  - `waitpid()`: wait for termination of a specific child

Process Termination

- `exit(int status)`
- Cleanups the process and returns to the kernel
- The status is passed to the parent
- Some operating systems also terminate child processes when parent terminates
  - « Cascading termination »

Practical Work 3: mini shell

- Write a new program in C
  - Name it « 03.practical.work.shell.c »
  - Execute commands entered from keyboard
  - Exit shell if user types `quit`
- Push your C program to corresponding forked Github repository