Multi Processing

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

ICT Department, USTH

Review

Review

- Process
- Scheduling
- IO Redirection

Process

- What is process?
- Process vs program?

Process

- Process is a program in execution state

Process

- Process is a program in execution state (**active**)

Process

- Process is a program in execution state (**active**)
- Why process?
    - Program is passive
    - No execution $\rightarrow$ what's running?

Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution → what's running?
- A process execution state contains

Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution $\rightarrow$ what's running?
- A process execution state contains
  - Processor state (context)

Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution $\rightarrow$ what's running?
- A process execution state contains
  - Processor state (context)
  - File descriptors

Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution → what's running?
- A process execution state contains
  - Processor state (context)
  - File descriptors
  - Memory allocation

Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution → what's running?
- A process execution state contains
  - Processor state (context)
  - File descriptors
  - Memory allocation
    - Process stack

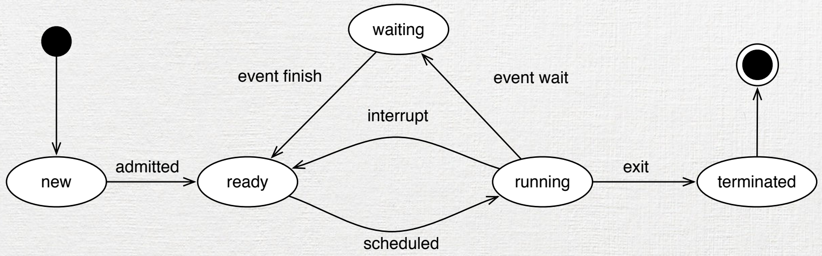
Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution → what's running?
- A process execution state contains
  - Processor state (context)
  - File descriptors
  - Memory allocation
    - Process stack
    - Data section

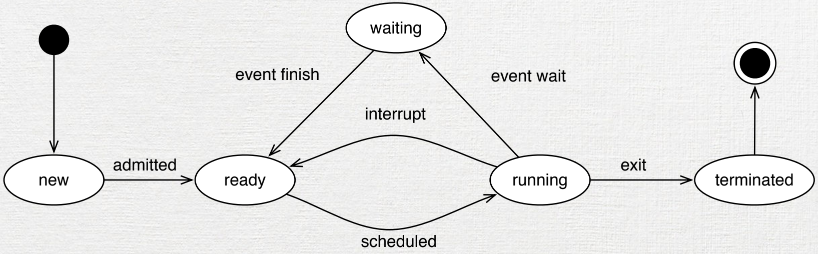
Process

- Process is a program in execution state (**active**)
- Why process?
  - Program is passive
  - No execution → what's running?
- A process execution state contains
  - Processor state (context)
  - File descriptors
  - Memory allocation
    - Process stack
    - Data section
    - Heap

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 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 Windows

```
BOOL WINAPI CreateProcess(
  _In_opt_    LPCTSTR               lpApplicationName,
  _Inout_opt_ LPTSTR                lpCommandLine,
  _In_opt_    LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_    LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_        BOOL                  bInheritHandles,
  _In_        DWORD                 dwCreationFlags,
  _In_opt_    LPVOID                lpEnvironment,
  _In_opt_    LPCTSTR               lpCurrentDirectory,
  _In_        LPSTARTUPINFO         lpStartupInfo,
  _Out_       LPPROCESS_INFORMATION lpProcessInformation
);
```

Source: MSDN

Process Creation on Windows

- A simplified WinAPI function:

```
UINT WINAPI WinExec(
  _In_ LPCSTR lpCmdLine,
  _In_ UINT   uCmdShow
);
```

Process Creation on Windows

- A simplified WinAPI function:

```
UINT WINAPI WinExec(
  _In_ LPCSTR lpCmdLine,
  _In_ UINT   uCmdShow
);
```

- It's deprecated.

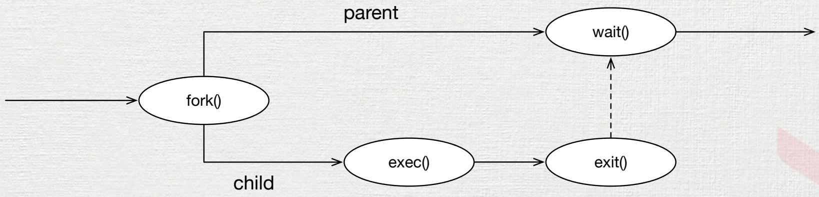
Source: MSDN

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

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?

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?

```
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

```c
#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;
}
```

Scheduling

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

Scheduling

- Maximize CPU usage
- Responsiveness for User interface
- Provide computational power for heavy-workload processes
- «Multitasking»
- Different characteristics of processes
  - CPU bound: spends more time on computation
  - I/O bound: spends more time on I/O devices (reading/writing disk, printing...)

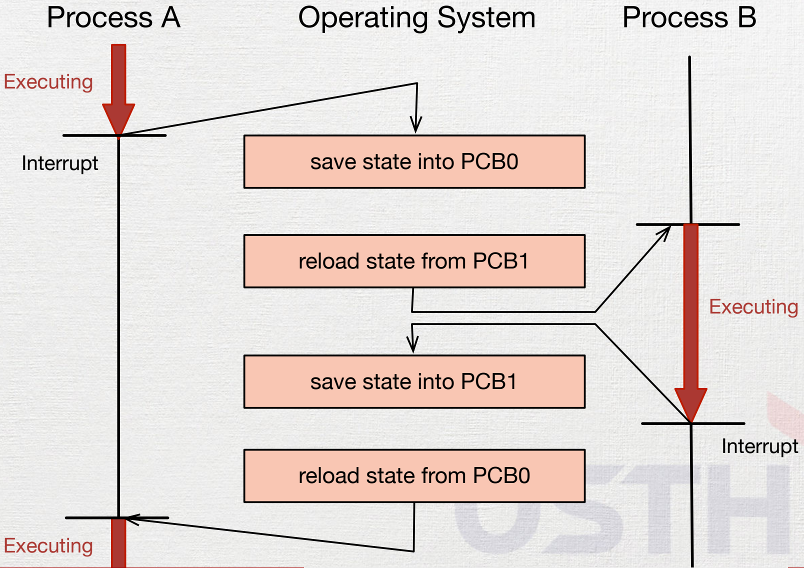
Scheduling

- By the ability to pause running processes
  - Preemption: OS forcely pauses running processes
  - Non-preemption (also cooperation): processes willing to pause itself

Scheduling

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

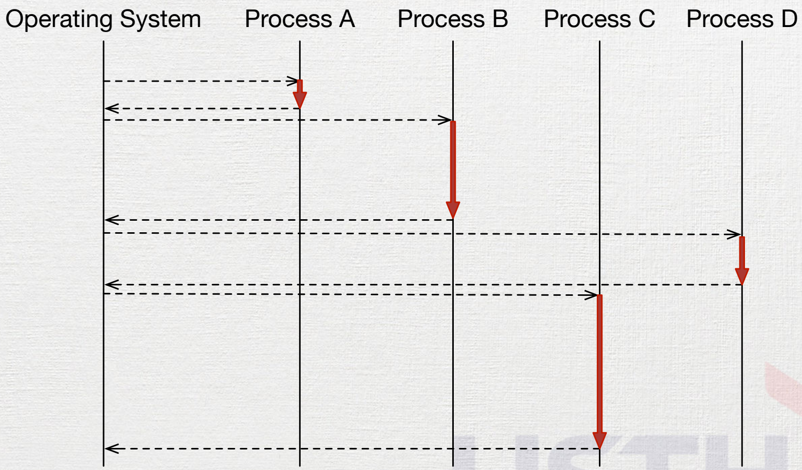
Scheduling with Context Switch



Scheduling with 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 with Context Switch



Scheduler

- Knowns
  - List of processes
  - Process states
  - Accounting information

Scheduler

- Knowns
    - List of processes
    - Process states
    - Accounting information
- Constraints
    - Process priority (if any)
        - Processes have scheduling **priority**
        - Indicates the importance of each process
        - Higher priority: more likely to be scheduled

Scheduler

- Problems
  - P1: What processes to run next?

Scheduler

- Problems
  - P1: What processes to run next?
  - P2: How long should it run?

Scheduler

- Problem 1: What processes to run next?
    - Job queue - set of all processes **entering** the system, stored on disk
    - Ready queue - set of all processes residing in **main memory**, ready and waiting to execute
    - Device queues - set of processes waiting for **an I/O device**
    - Lists of PCBs
    - Processes change state $\rightarrow$ they migrate among the various queues

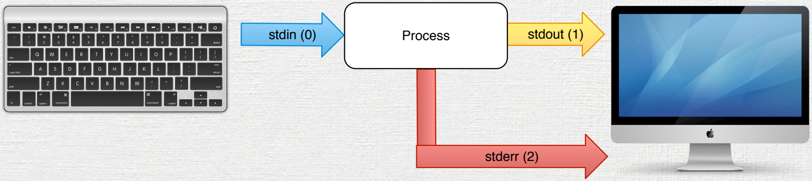
Scheduler

- Problem 2: How long should it run?
  - First In First Served
  - Earliest Deadline First
  - Shortest Remaining Time
  - Round Robin
  - ...

Scheduler

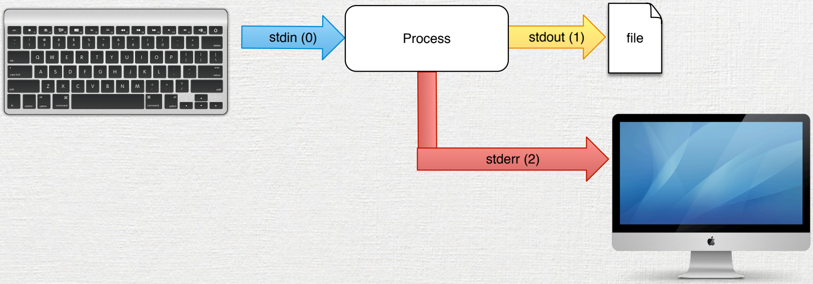
Algorithm	Preempt?	Priority?	Note
First Come, First Served	No	No	Depends on arrival time
Shortest-Job-First	No	Yes	Low waiting time ω
Shortest-Remaining-Time-First	Yes	Yes	Preemptive SJF, low ω
Round Robin	Yes	No	Low response time ρ
Multilevel Queue	Depends	Depends	Several subqueues, permanent
Multilevel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

IO Redirection



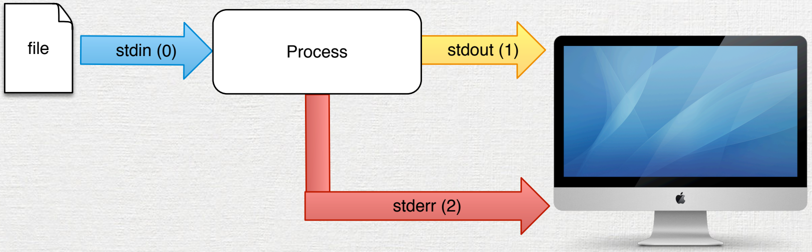
Default: input from keyboard and output to terminal

IO Redirection



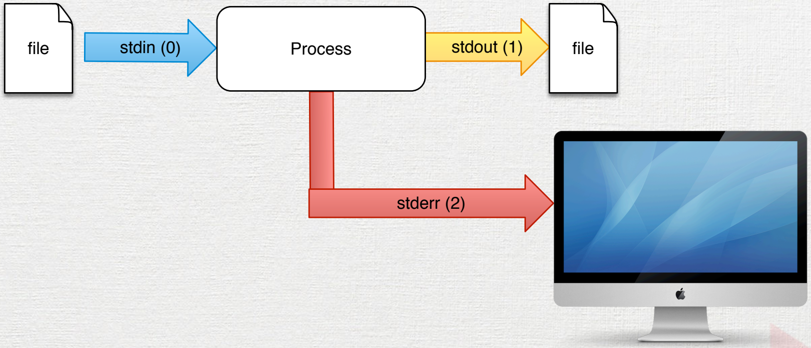
Input from keyboard and output to file

IO Redirection



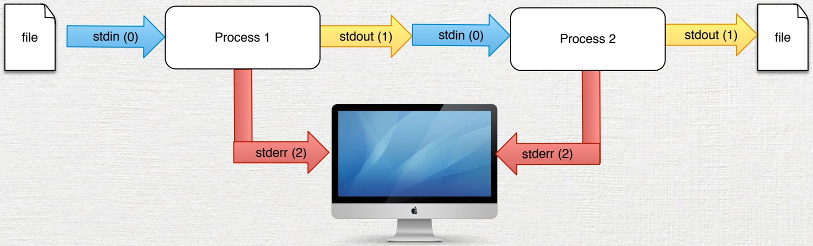
Input from file and output to terminal

IO Redirection



Input from file and output to another file

IO Redirection



Input from file, pipe output of Process 1 to Process 2, output to another file

Processes

Modules

- `os`
- `subprocess`

Task

- Create a process
    - Run and wait for finish
    - Run in background
    - Run with timeout
- IO redirection
    - Redirect input
    - Redirect output
    - Redirect with pipe
- Terminate
- Get return code

os module

- os module is deprecated in Python 3
- This is for references only.

Task	How
Run and wait	`os.system("ps aux")`
Run in background	`os.system("long_command.sh &")`
Timeout	N/A
Redirect input	`os.popen("bc", "w").write("1+2")`
Redirect output	`print(os.popen("ps aux", "r").readlines())`
Redirect with pipe	`os.pipe(), os.fork()`
Terminate	`os.kill(pid, signal.SIGTERM)`
Get return code	return value of `os.system()`

`subprocess` module

Task	How
Run and wait	`subprocess.run(["ps", "aux"])`
Run in background	`subprocess.Popen("long_command.sh")`
Timeout	`subprocess.run("long_command.sh", timeout = 10)`
Redirect input	`subprocess.Popen("bc", stdin=subprocess.PIPE).communicate(b"3+4\n")`
Redirect output	`subprocess.Popen(["ps", "aux"], stdout=subprocess.PIPE).communicate()`
Redirect with pipe	`subprocess.Popen("bc", stdin=anotherProcess.stdout)`
Terminate	`anotherProcess.terminate()`, `anotherProcess.kill()`
Get return code	`subprocess.check_output()`, catch `CalledProcessError`

Practice!

Practical work 7: Python shell

- Create a new python program, name it «7.shell.py»
- Make a shell
  - User inputs command
  - Shell executes the command, print output
  - Support IO redirection
    - input from file to process
    - output from process to file
    - e.g. input from one process being output of another

Practical work 7: Python shell

- Run it and test some commands
  - `ls -la`
  - `ls -la > out.txt`
  - `bc < input.txt`
  - `ps aux | grep term`
- Push your work to corresponding forked Github repository