Process

COMP3230B Workshop 2

Learning Outcomes

- To learn the basics of multiprocess programming
- To understand the flow of Assignment 1

Workshop Setup

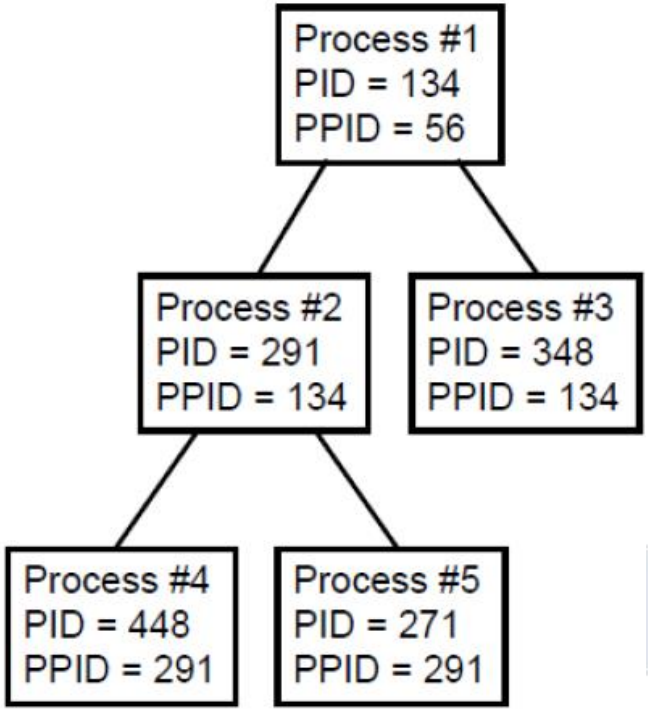
- **Time: 13:30 – 15:20**

- **In-Class Preparation**
  - **Boot the Linux system or VM in your laptop, or;**
  - **Connect CSVPN and log in X2Go**

Multiprocess Programming

Linux Process

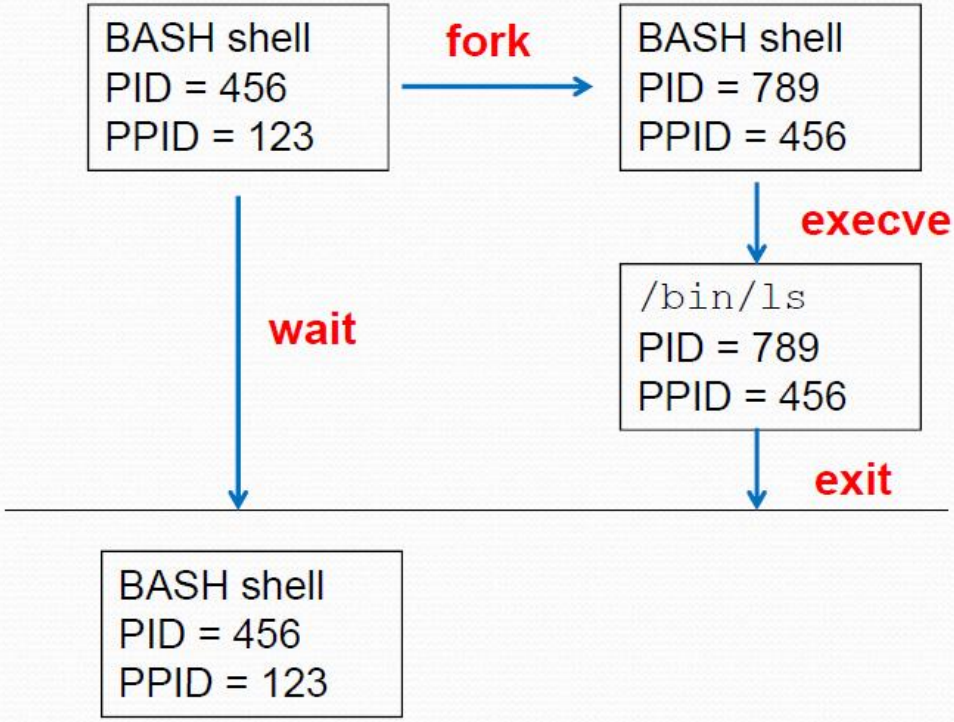
- Each process has a unique Process ID (**PID**) that allows the kernel to identify it.
- Each process may start an unlimited number of other processes called child processes.
- Each process has only one parent process.
- Each process must have been started by an existing process called a parent process.
- Each process has a Parent Process ID (**PPID**) which identified the process that started it.
- PID is generated randomly from free entries in a process table used by the Linux kernel.



Relationship between Parent and Child Processes

Process Creation and Termination

- How to start a process?
  - E.g. executing `ls` through a shell



fork()

- System call to create a new process (child) by duplicating the caller process.
- Format: pid_t fork();
- Each process receives different return value from fork().
  - In the caller process, the return value is the PID of the child process.
  - In the child process, the return value is 0.
- Two processes will run "concurrently"
  - In the child process, the execution starts at the location after fork().

fork.c

E.g. fork.c

```
#include <stdio.h>
#include <sys/types.h>

int main () {
  pid_t pid=fork();

  if (pid < 0) {
    printf("can't create process\n");
  } else if (pid == 0) {
    printf("I am child\n");
  } else {
    printf("I am parent\n");
  }
  return 0;
}
```

Output:

```
I am parent
I am child
```

fork.c

Parent Process

```
pid_t pid = fork();
if (pid < 0) {
  printf("can't create process\n");
}
else if (pid == 0) {
  printf("I am child\n");
}
else {
  printf("I am parent\n");
}
...
```

a new process is created

Child Process

```
pid_t pid = fork();
if (pid < 0) {
  printf("can't create process\n");
}
else if (pid == 0) {
  printf("I am child\n");
}
else {
  printf("I am parent\n");
}
...
```

getpid() / getppid()

- System call
- Function: get `pid` and `ppid` of the current process
- Format:
  - `pid_t getpid();`
  - `pid_t getppid();`
- Returned type: `pid_t`
  - Treat it as integer

```
pid_t my_pid, parent_pid;


my_pid = getpid();

parent_pid = getppid();

printf("My pid is %d\n", my_pid);

printf("My parent's pid is %d\n", parent_pid);
```

Example: Zombie

- Download zombie.c
- Compile and run
- In a new terminal:
  - $ ps aux | grep <user name>
- Zombie is not actually existing any more. Its record is kept until parent reads it.
  - Zombie processes only consume some space in process table. The process with Z+ status
- The child finishes, but the parent is in deep sleep
- The entry in the process table is kept until the parent wakes up and handles the exit message of child

exec family

- exec is actually family: 6 functions.
  - Only *execve* is *system call*.
  - Other five are library call that calls *execve*.
  - Example:

```
execlp("/bin/ls", "ls", NULL);
```

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
```

- As soon as a program (process) call **exec**, most of its states will be **replaced** by the new command-process, except a few information, such as PID.

Text segment | Data segment | stack | .....

We will try this in Exercise

wait()

- Function: used by parent to wait for a child process to terminate.
  - Can't wait for a process which is not started by the caller.
- Format:
  - `pid_t wait(int *status);`
  - `pid_t waitpid(pid_t pid, int *status, int options);`
    - *pid_t* means process id.
    - *status* is used to store the child's status information, such as whether the child terminated normally.
    - *options*: WNOHANG /WUNTRACED
- Returned value: process id of child, or -1 for error.
  - What error? Such as:
    - What if the process specified in pid does not exist.
    - What if the options argument was invalid.
  - Parent (the caller) wait until child exit.
  - If you don't care the status, you can use *NULL* for simplicity
    - E.g. `waitpid(123, NULL, 0);`

What should be included as header??

- **To view manual pages, issue commands in terminal**
  - **man <optional section number> <manual page>**
  - **e.g.** man fork**,** man 3 printf
- **There may be manual pages with the same name but different section numbers, e.g. printf(1), printf(3)**
  - **1 for commands, 2 for system calls, 3 for library calls, ...**
- **To search for pages, use -k option.**
  - man -k sort

Navigation

- **Similar to navigation in Vi editor**
  - j **(or** down-arrow**) for one line down,** k **(or** up-arrow**) for one line up**
  - Ctrl-v **for next page,** Ctrl-b **for previous page**
  - xxG **for xx-th line, where xx is integer.**
  - /abc **for searching "abc", then n for next match, and N for previous match**
  - h **for help,** q **for quit**

Keywords in Manual Pages

- **Synopsis**
  - **What headers should I include?**
  - **What are the parameters?**
- **Description**
  - **What does the function do?**
- **Return Values, Errors**
  - **How do I know the execution result?**
- **See Also**
  - **What are the related functions?**
- **Standards**
  - **Is it still useful on another OS?**

Example: Make Use of Linux Command

- In fork.c (Task 1 of Exercise)
  - Let child exec the ls command (As shown in Page 12)
  - Let parent wait for child

- In this way, programmers can use Linux commands as part of their programs.
  - e.g. Use *sort* to sort an input -> wait -> proceed to read the sorted file

Assignment 1

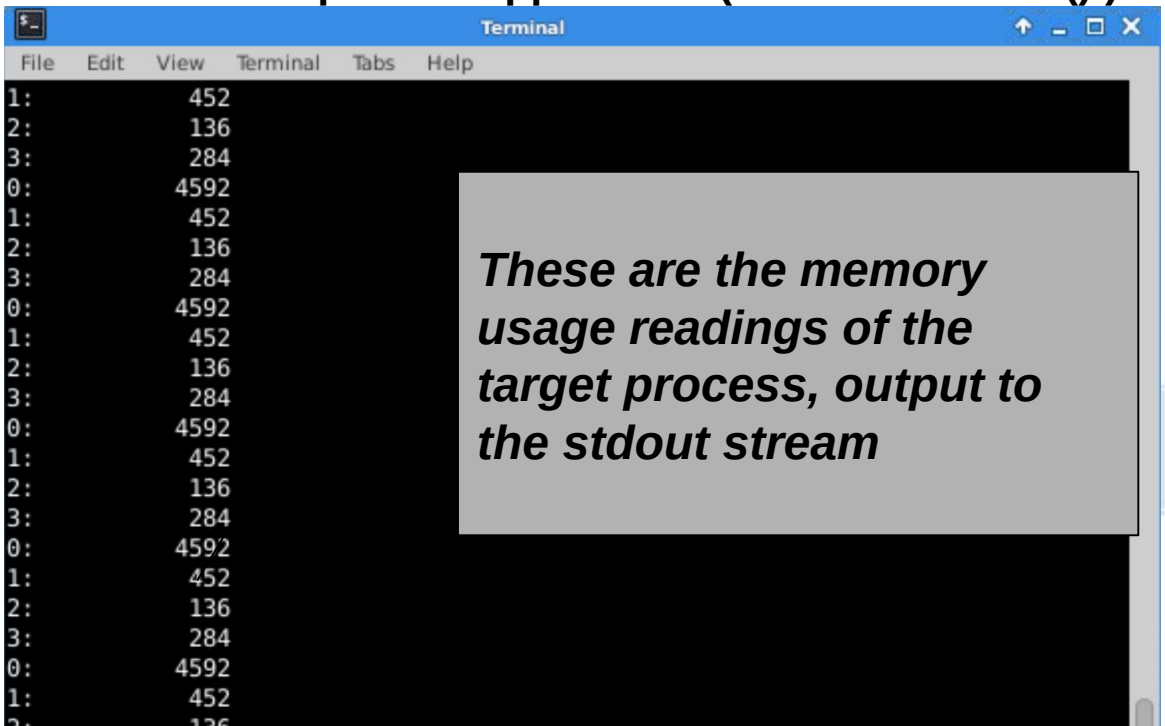
Extend tmem

- **Monitor the run-time memory usage**
  - **To access the total memory information about a process on Linux, we use the /proc virtual file system. Within it there is a directory full of information for each active process id (pid). By reading /proc/(pid)/status we can obtain information about memory.**

- **Credit to Lockless**
  - **http://locklessinc.com/articles/memory_usage/**

tmem_template.c

- **The given template is a runnable program (can be found under Assignment 1)**
  - **gcc tmem_template.c –o tmem**
  - **./tmem null**
    - **null is a built-in pseudo application (as shown in null())**



These are the memory usage readings of the target process, output to the stdout stream

Realtime Display

- A perl script that drives gnuplot
  - Read labeled streams from stdout

Pass stdout of process 1 to stdin of process 2

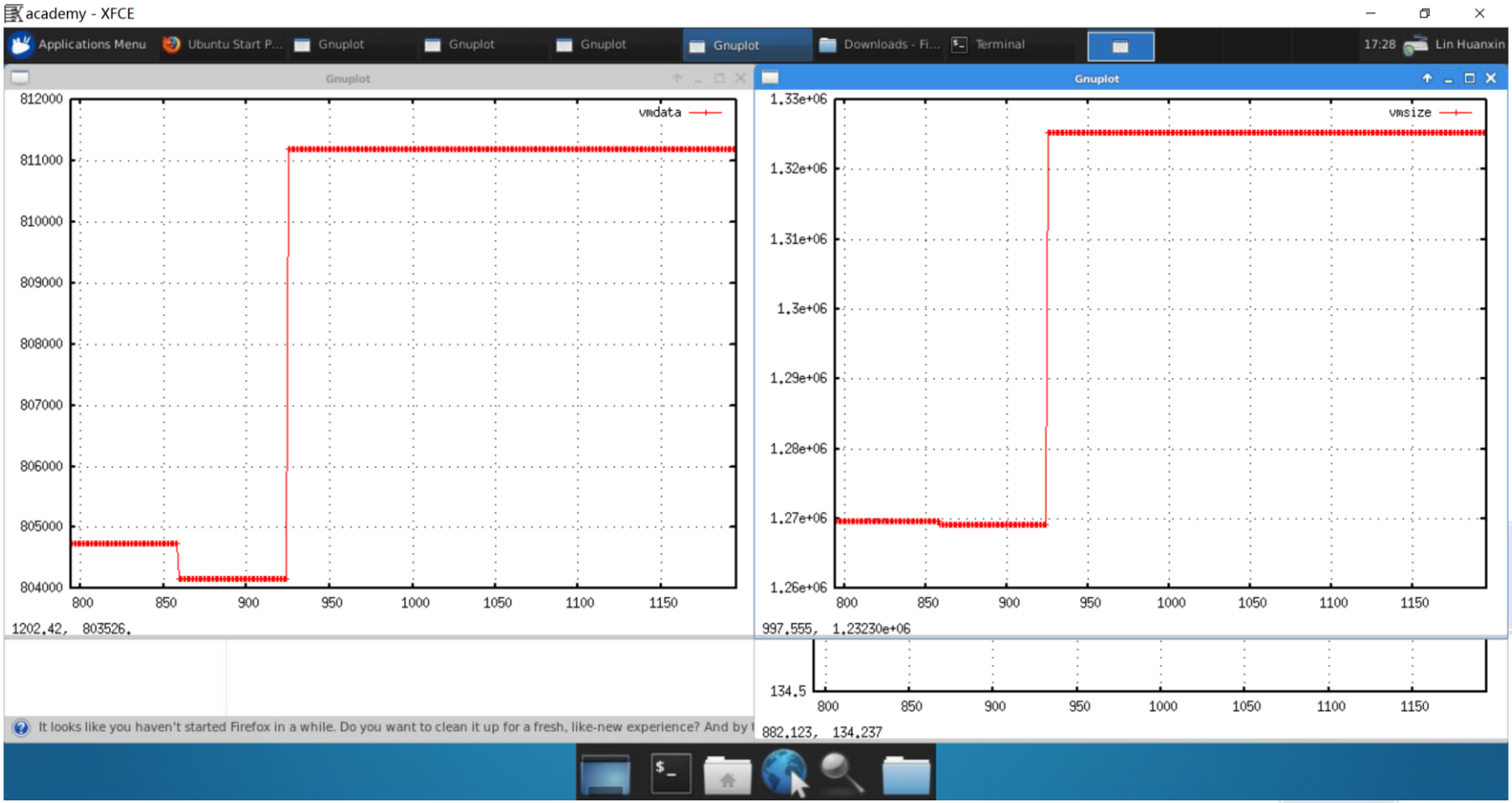
- Usage
  - ./tmem firefox | perl driveGnuPlots.pl 4 400 400 400 400 vmsize vmdata vmstk vmrss
    - Before you try, close all the Firefox windows!

- Credit to ttsiodras
  - http://users.softlab.ntua.gr/~ttsiod/gnuplotStreaming.html

Running on own machine

- **Install gnuplot and gnuplot-x11**
- **Ubuntu**
  - **sudo apt-get install gnuplot gnuplot-x11**
- **Fedora**
  - **sudo yum install gnuplot gnuplot-x11**

Realtime Display



main()

```
//Set the signal handler function as sig_chld
act.sa_handler = sig_chld;

/* We don't want to block any other signals */
sigemptyset(&act.sa_mask);

act.sa_flags = SA_NOCLDSTOP;

if (sigaction(SIGCHLD, &act, NULL) < 0)
{
    fprintf(stderr, "sigaction failed\n");
    return 1;
}
```

- **Set signal handler as sig_chld**
  - **Handler means a function that is called every time its corresponding signal arrives**
- **SIGCHLD**
  - **A signal that a child sends to parent when child finishes**
  - **We may also send through the terminal**

sig_chld()

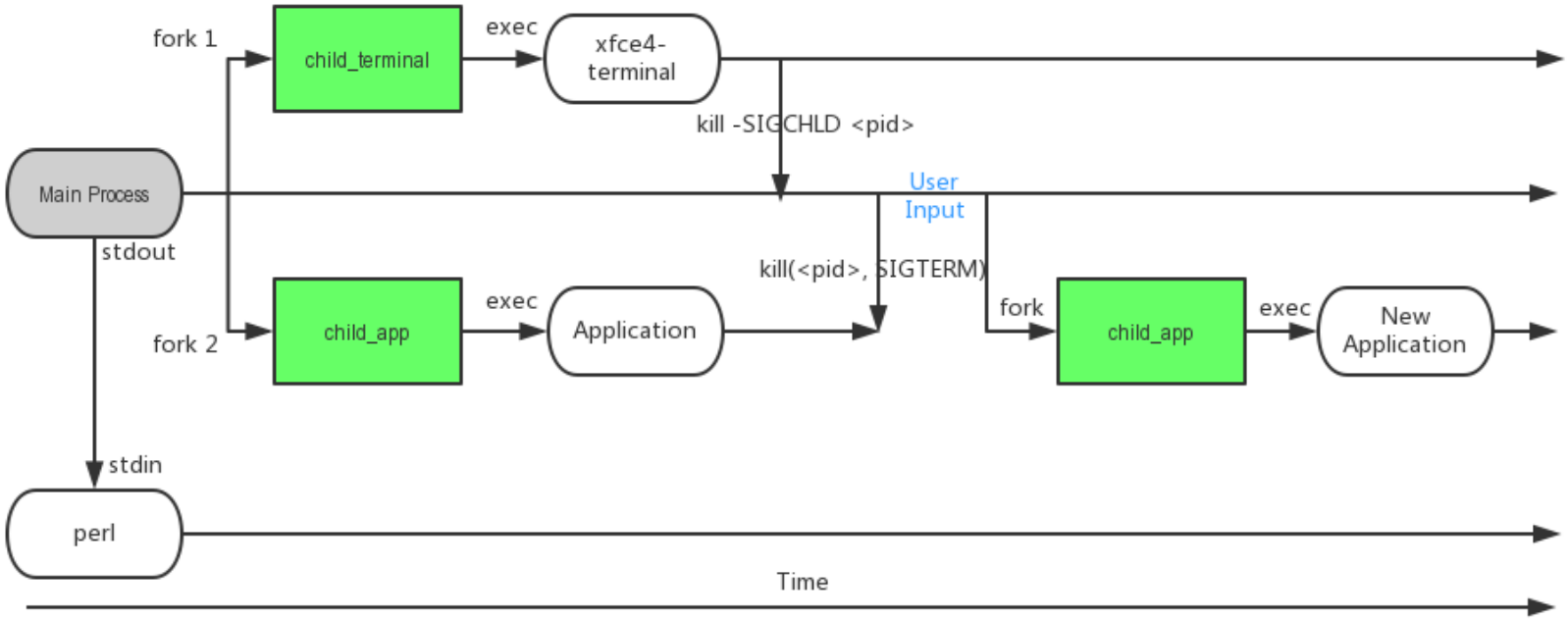
```
pid = waitpid(-1, &status, WNOHANG);
if (pid < 0)
{
    fprintf(stderr, "waitpid failed\n");
    return;
}

/* pid blocked */
if(pid == block_pid)
    return;

/* signal not from children */
if (pid != target_pid )
```

- **WNOHANG: Enter handler immediately without looking at the sender pid**
- **Only terminate when direct child terminates**

At completion of assignment



Exercise

Workshop Exercise 2

- **Task 1: Finish the task on Page 17 and submit fork.c**
  - **Exec the ls command. Not sort.**

- **Task 2: Submit 1 screenshot showing the running of tmem with real-time display**
  - **We just want to make sure everyone can run the basic program**
  - **./tmem firefox | perl driveGnuPlots.pl 4 400 400 400 400 vmsize vmdata vmstk vmrss**

Workshop Exercise 2

- **Modify example_1.c to make sure the main process creates exactly 4 child processes**

```
// wrong example_1.c
#include <unistd.h>    // for fork()
#include <stdio.h>

int main() {
  fork();
  printf("Child 1 created\n");
  fork();
  printf("Child 2 created\n");
  fork();
  printf("Child 3 created\n");
  fork();
  printf("Child 4 created\n");
  return 0;
}
```

- **Submit the corrected example_1.c and 1 screenshot showing the correct output (output order does not matter)**