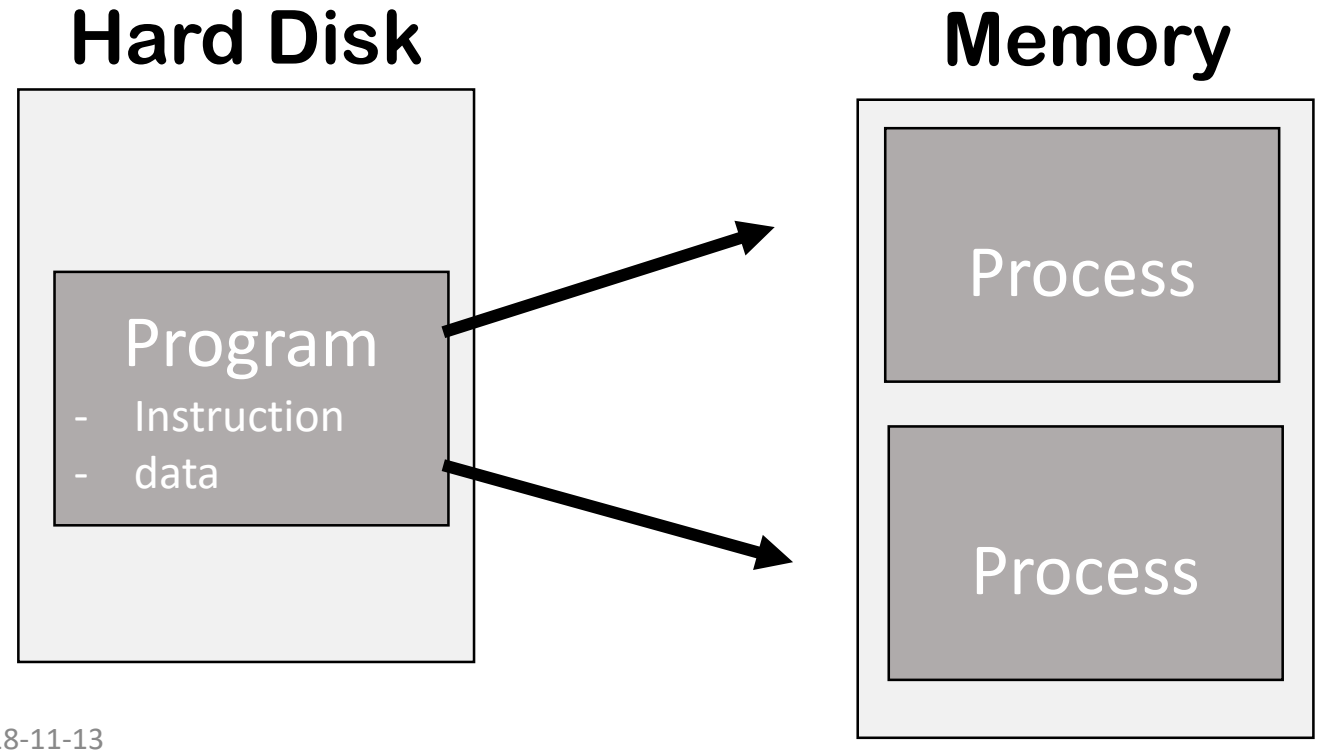
Lab9:
Process & Signal

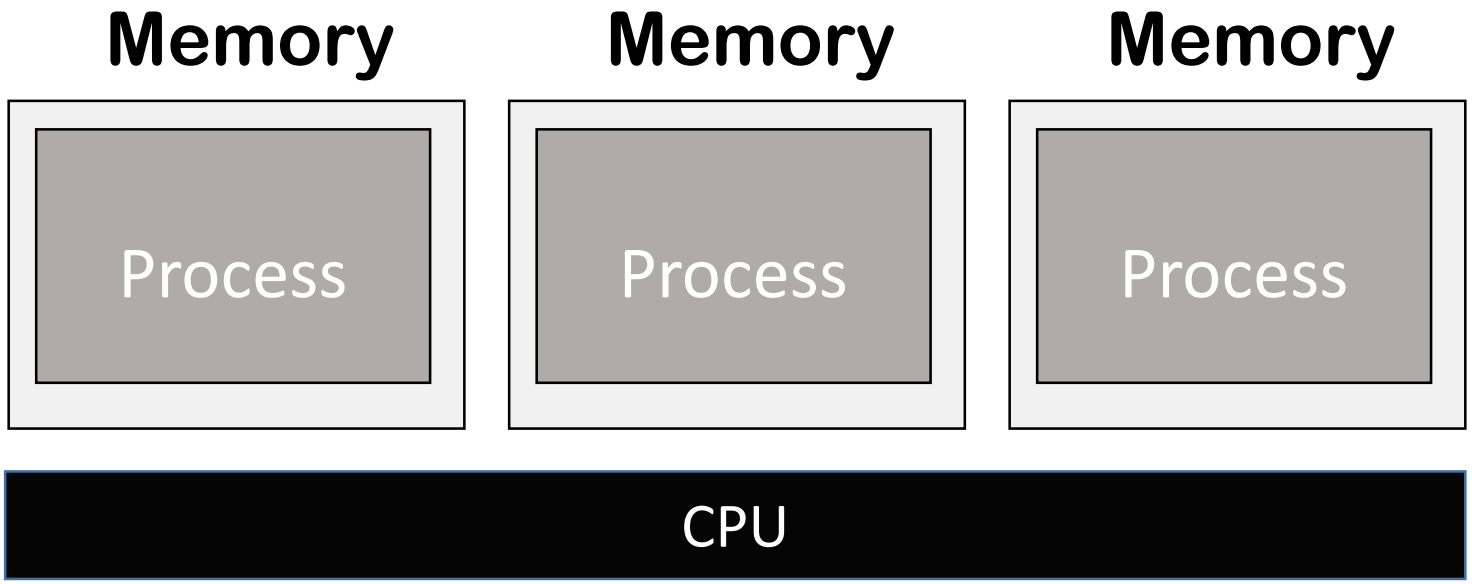
Process (1/3)

- **An instance of a computer program that is <u>being executed</u>**
  - **Definition of "Process": an instance of a running program**



Process (2/3)

- Provide each program with two key abstractions:
  - Logical control flow – exclusive use of the CPU
    - Context switching
  - Private address space – exclusive use of main memory
    - Virtual memory



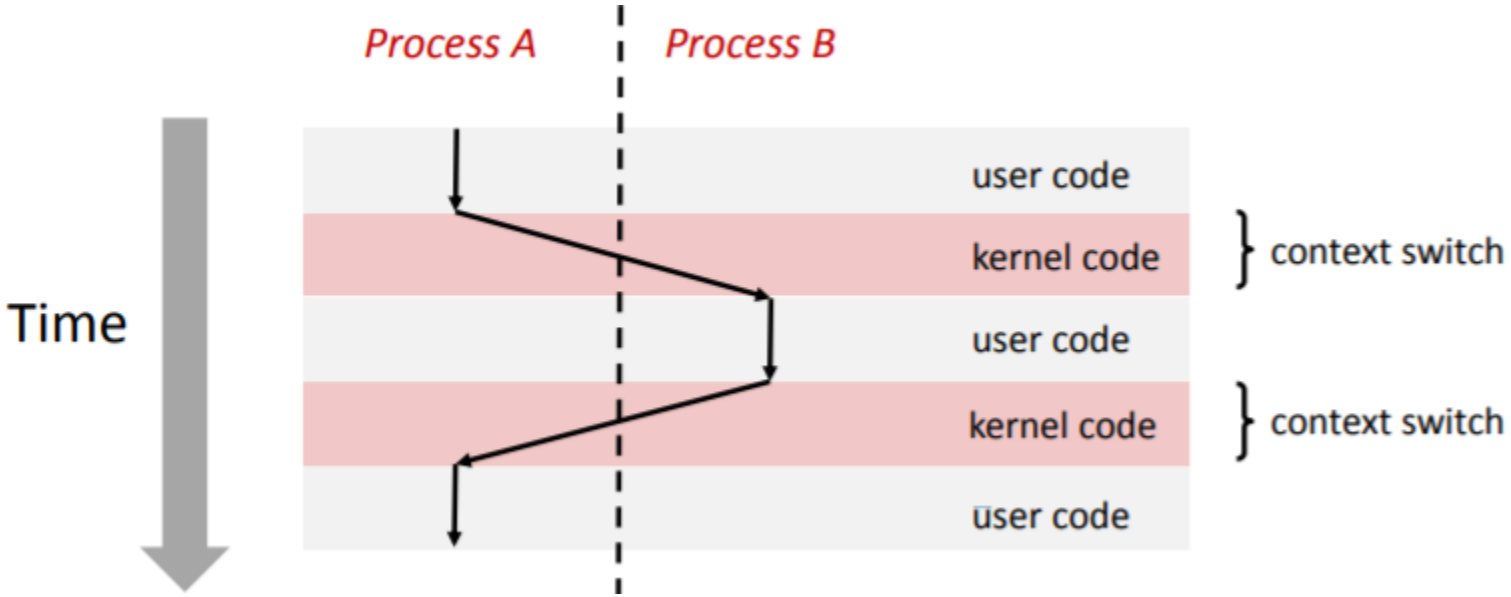
Process mode

- User mode
  - A mode for user operation
  - Various commands are executed with user authority by operation such as arithmetic operation.

- Kernel mode
  - A mode for accessing memory and hard disk, which are resource of a computer, and is executed with kernel privilege.

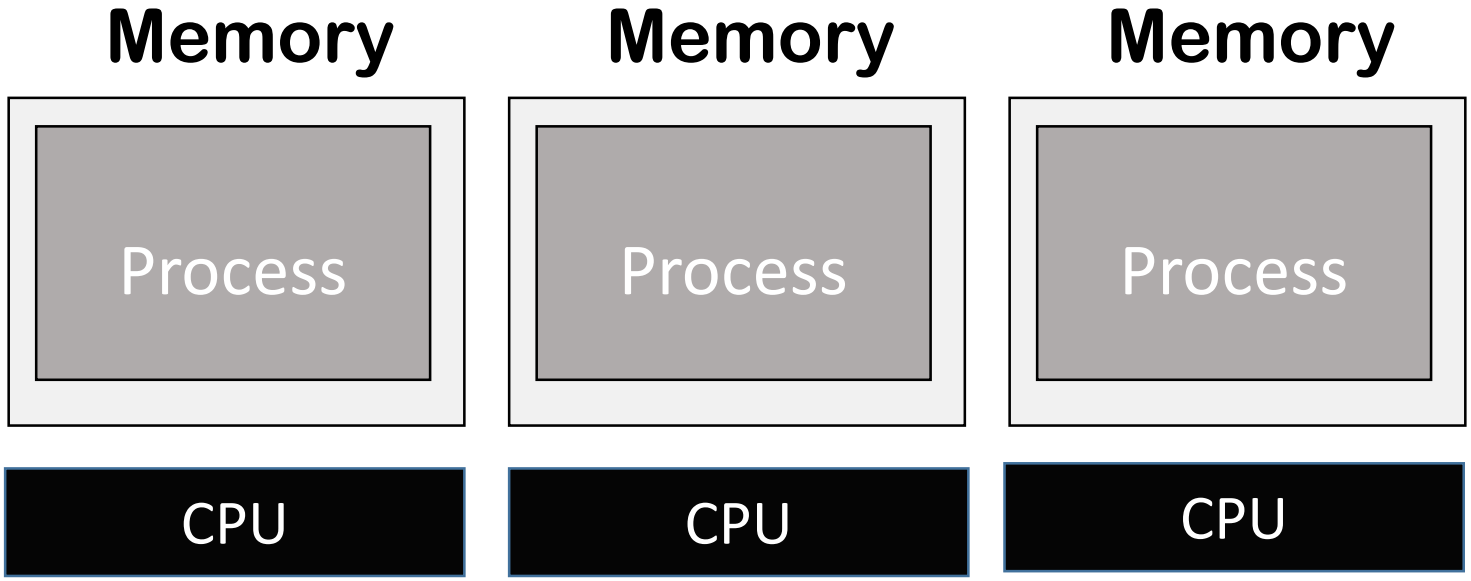
Context Switching

- Process are managed by a shared chunk of OS code called the kernel.
  - [Important] the kernel is not a separate process, but rather runs as part of some user process.



Process (2/3)

- Provide each program with two key abstractions:
  - Logical control flow
    - Context switching
  - Private address space
    - Virtual memory



Process (3/3)

- Each process is identified with PID and name.
  - name : Process name
  - PID : Process ID, unique number
  - PPID : Parent process PID
  - PGID : Process Group ID

Process state

- Running
  - Process is either executing, or waiting to execute.
- Stopped
  - Process execution is suspended and will not be scheduled until further notice.
- Terminated
  - Process is stopped permanently.
- Zombie
  - Execution is completed, but remain in memory.

Process Execution – execl

```
#include <unistd.h>
int execl(const char *path, const char *arg, ...);
```

- Execute a new process and overwrite the current process
- Path: the path to the program to be executed
- arg: execution arguments
  - Must enter NULL at the end

Execution Example

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv)
{
  printf("Start\n");
  execl("/bin/ls", "ls", "-al", NULL);
  printf("End\n");
  return 0;
}
```

- result

```
namgyu@HPC:~$ ./execTest
Start
total 124
drwxr-xr-x 16 namgyu namgyu  4096 Nov 13 10:47 .
drwxr-xr-x  6 root   root    4096 Sep 19 15:28 ..
drwxrwxr-x  2 namgyu namgyu  4096 Sep 30 22:52 archi_Pro
-rw-------  1 namgyu namgyu 16505 Nov 11 15:13 .bash_history
-rw-r--r--  1 namgyu namgyu   220 Feb 27  2018 .bash_logout
-rw-r--r--  1 namgyu namgyu  3637 Feb 27  2018 .bashrc
drwx------  5 namgyu namgyu  4096 Mar 13  2018 .cache
drwx------  5 namgyu namgyu  4096 May 28 19:19 .config
drwxrwxr-x  3 namgyu namgyu  4096 Nov  1 20:55 CSED211
drwxr-xr-x  3 namgyu namgyu  4096 Mar 13  2018 Desktop
drwxrwxr-x  3 namgyu namgyu  4096 Nov  4 20:42 evaluate_p2
drwxrwxr-x 68 namgyu namgyu  4096 Nov  4 15:50 evaluate_p3
-rwxrwxr-x  1 namgyu namgyu  8613 Nov 13 10:46 execTest
-rw-rw-r--  1 namgyu namgyu   174 Nov 13 10:46 exectest.c
drwx------  3 namgyu namgyu  4096 Mar 13  2018 .gnome
drwxrwxr-x  3 namgyu namgyu  4096 Mar 13  2018 .local
drwx------  5 namgyu namgyu  4096 Mar 13  2018 .mozilla
drwx------  3 namgyu namgyu  4096 Mar 13  2018 .pki
-rw-r--r--  1 namgyu namgyu   675 Feb 27  2018 .profile
drwx------  2 namgyu namgyu  4096 Feb 27  2018 .ssh
drwxr-xr-x  2 namgyu namgyu  4096 Sep 30 22:47 .vim
-rw-------  1 root   root    5191 Sep 30 22:51 .viminfo
drwxr-xr-x  2 namgyu namgyu  4096 May 28 21:38 .wireshark
-rw-------  1 namgyu namgyu   245 Nov 13 10:45 .Xauthority
namgyu@HPC:~$
```

Process Creation – fork

```
#include <unistd.h>

pid_t fork(void);
```

- Copy the parent process
- Return 0 to the child process, child's PID to parent process
- Child almost identical to parent
  - Not PID, virtual address space

Fork Example

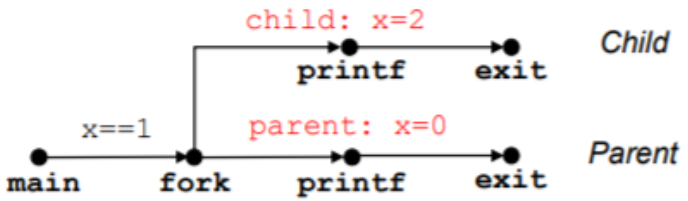
```
int main(int argc, char** argv)
{
    pid_t pid;
    int x = 1;

    pid = Fork();
    if (pid == 0) {  /* Child */
        printf("child : x=%d\n", ++x);
        return 0;
    }

    /* Parent */
    printf("parent: x=%d\n", --x);
    return 0;
}
```

fork.c

- **Call once, return twice**
- **Concurrent execution**
  - Can't predict execution order of parent and child




```
linux> ./fork
parent: x=0
child : x=2
```

```
linux> ./fork
child : x=2
parent: x=0
```

```
linux> ./fork
parent: x=0
child : x=2
```

```
linux> ./fork
parent: x=0
child : x=2
```

Other Functions

- void exit(int status)
  - Terminate process
  - Convention: normal return status is 0, nonzero is error
- int wait(int *child_status)
  - Synchronize with children
- Etc…
- If you want to know more,
  you can find in textbook or Google!

Signal (1/3)

- A signal is a small message that notifies a process that an event of some type has occurred in the system.
  1. To alarm an asynchronous event
  2. To synchronize events
- Use [kill –l] command, check out the signal list

```
namgyu@HPC:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Signal (2/3)

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

int kill(pid_t pid, int sig);
```

Using some condition, send the signal to specific process or group
pid > 0
pid == 0
pid == -1
pid < -1

#kill -signal pid

ex) kill –SIGKILL 100
ex) kill –9 100

Ctrl + C → SIGINT
Ctrl + Z → SIGSTP

SIGKILL	Kill the process
SIGALARM	Timer signal
SIGSTP	Stop the process
SIGCONT	Restart the process
SIGINT	Interrupt the process
SIGSEGV	Segmentation violation

Signal (3/3)

- Kill function only uses to send signal.
- By using sighandler_t signal function, it can catch the signal.

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- signum : signal number to control
- sighandler_t : when signum is received, the function to call

Signal Example

```c
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

void sig_handler(int signo);

int main(int argc, char **argv)
{
     int i = 0;
     signal(SIGINT, (void *)sig_handler);
     while(1)
     {
        printf("%d\n", i);
        sleep(2);
        i++;
     }
}

void sig_handler(int signo)
{
      printf("I Received SIGINT(%d)\n", SIGINT);
}
```

```
namgyu@HPC:~$ ./a.out
0
1
2
^CI Received SIGINT(2)
3
4
^CI Received SIGINT(2)
5
6
7
8
9
10
^CI Received SIGINT(2)
11
^Z
[3]+  Stopped                 ./a.out
namgyu@HPC:~$
```

Practice

1. Find the SIGCHLD signal
   -https://docs.oracle.com/cd/E19455-01/806-4750/signals-7/index.html
2. Using the fork function, make the 3 child processes (print its PID)
3. Using sleep(n) function, block the 3 child processes
4. After that, using the exit function, terminate all child processes
5. Catch the SIGCHLD signal, and print it.

Maybe you can solve this practice easily when you use the waitpid function.