Recitation 8

TA Hanxiong Chen
hc691@rutgers.edu

Signal

User program can invoke OS services by using system calls.

What if the program wants the OS to notify it *asynchronously* when some event occurs?

Signals

- UNIX mechanism for OS to notify a user program when an event of interest occurs
- Potentially interesting events are predefined: e.g., segmentation violation, message arrival, kill, etc.
- When interested in “handling” a particular event (signal), a process indicates its interest to the OS and gives the OS a procedure that should be invoked in the upcall

Signal (cont.)

- When there is a signal?
  - System send a signal such as SIGSEGV
  - User send a signal from cmd line such as SIGINT (ctrl + c)
  - Sent by other program by calling kill() system call
  - Sent by program itself
  - ...
- How to use signal?
  - Step 1: write your signal handler function: void signal_handler(int signum){}
  - Step 2: bind your function with a signal: signal(SIGINT, signal_handler);

Signal_handler is a call_back function. That means once there is a signal you registered comes, the system would call the corresponding function (which you bind this signal with) to handle this case. The handler function must have void return type.

Signal Handler Example

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

void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("\n received SIGINT\n");
    exit(0);
}
```

```
int main(void)
{
  if (signal(SIGINT, sig_handler) ==
SIG_ERR)
        printf("\ncan't catch SIGINT\n");

  while(1)
        sleep(1);

  return 0;
}
```

Signal (cont.)

- How to send a signal to a process?
  - Use kill() system call in a program (send to calling program or other running processes)
  - Use kill command in terminal: kill 9 3213 (send SIGKILL to the process whose pid is 3213)

Eg. we can interrupt the process itself by doing:

```
int main() {
        ...
        signal(getpid(), SIGINT);
        ...
        return 0;
}
```

IPC (Inter-Process Communication)

IPC is a way to send messages among different processes. Commonly we have three ways:

- Pipe
- Shared Memory
- Signal

IPC (cont.)

- Pipe
  - pipe(int *): You need to pass a int array into the pipe system call.
  - int pp[2]; pipe(pp);
  - Only two elements in the array are enough.
  - pipe() returns -1 if failed. If success, pp will contains two file descriptors. pp[0] is for reading fd and pp[1] is for writing.
  - If you are using pipe with fork, make sure to do pipe first then do fork. If you do in a wrong order, you would not create a pipe between the parent and child processes.
  - One pipe can only make a one-to-one communication on one direction. (not two directions)

Why we cannot share one pipe with multiple processes?

Shared Memory

There are two common ways to do this:

1. shmget() from shm.h header
   a. https://users.cs.cf.ac.uk/Dave.Marshall/C/node27.html
   b. You can check the link for a sample code.
2. mmap() mapping memory
   a. This is to choose an unallocated memory area which between heap and stack to do the mapping.
   b. You can do mmap() by setting flag MAP_SHARED to make it a shared memory area.
   c. https://stackoverflow.com/questions/5656530/how-to-use-shared-memory-with-linux-in-c
3. Useful Link: http://man7.org/training/download/posix_shm_slides.pdf

Shared Memory

What can be wrong when using shared memory?

Synchronization issues!!

Make sure to use semaphore to make your shared memory can be visit only one time one process.

Demo

NOW DEMO TIME!!!!!!

References

1. http://man7.org/training/download/posix_shm_slides.pdf
2. https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/19/lec.html
3. http://www.csl.mtu.edu/cs4411.ck/www/NOTES/signal/kill.html
4. https://stackoverflow.com/questions/5656530/how-to-use-shared-memory-with-linux-in-c
5. https://users.cs.cf.ac.uk/Dave.Marshall/C/node27.html
6. http://menehune.opt.wfu.edu/Kokua/More_SGI/007-2478-008/sgi_html/ch03.html