CSCI3150 – Tutorial 6

ASSIGNMENT 1, PHASE II

Congrats, you have passed Phase 1

Now you have to finish phase 2....

Phase 2 : Scheduler 🕛

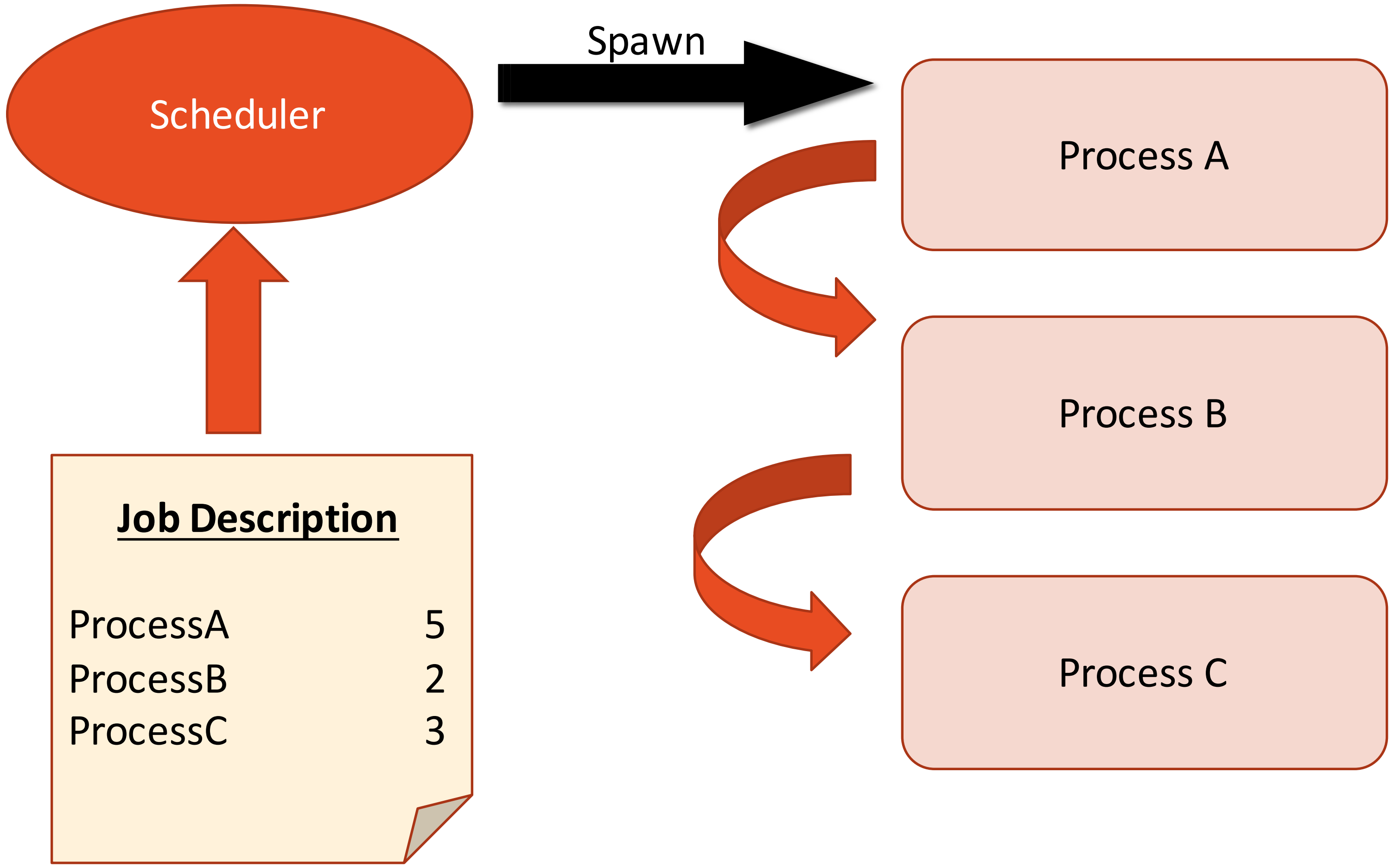
- a process to create a list of jobs and limit the running times of them.

First In First Out (FIFO Mode)

1. The scheduler reads the job description file (contains program name and duration).
2. It runs the job one by one, according to the order.
3. Scheduler counts the time:
    1. If the job ends **BEFORE** the scheduler, wake up the schedule.
    2. If time **EXCEEDS** the limit, the scheduler kills it.

4. Then next job will be run and same time counting applies.

First In First Out (FIFO Mode)



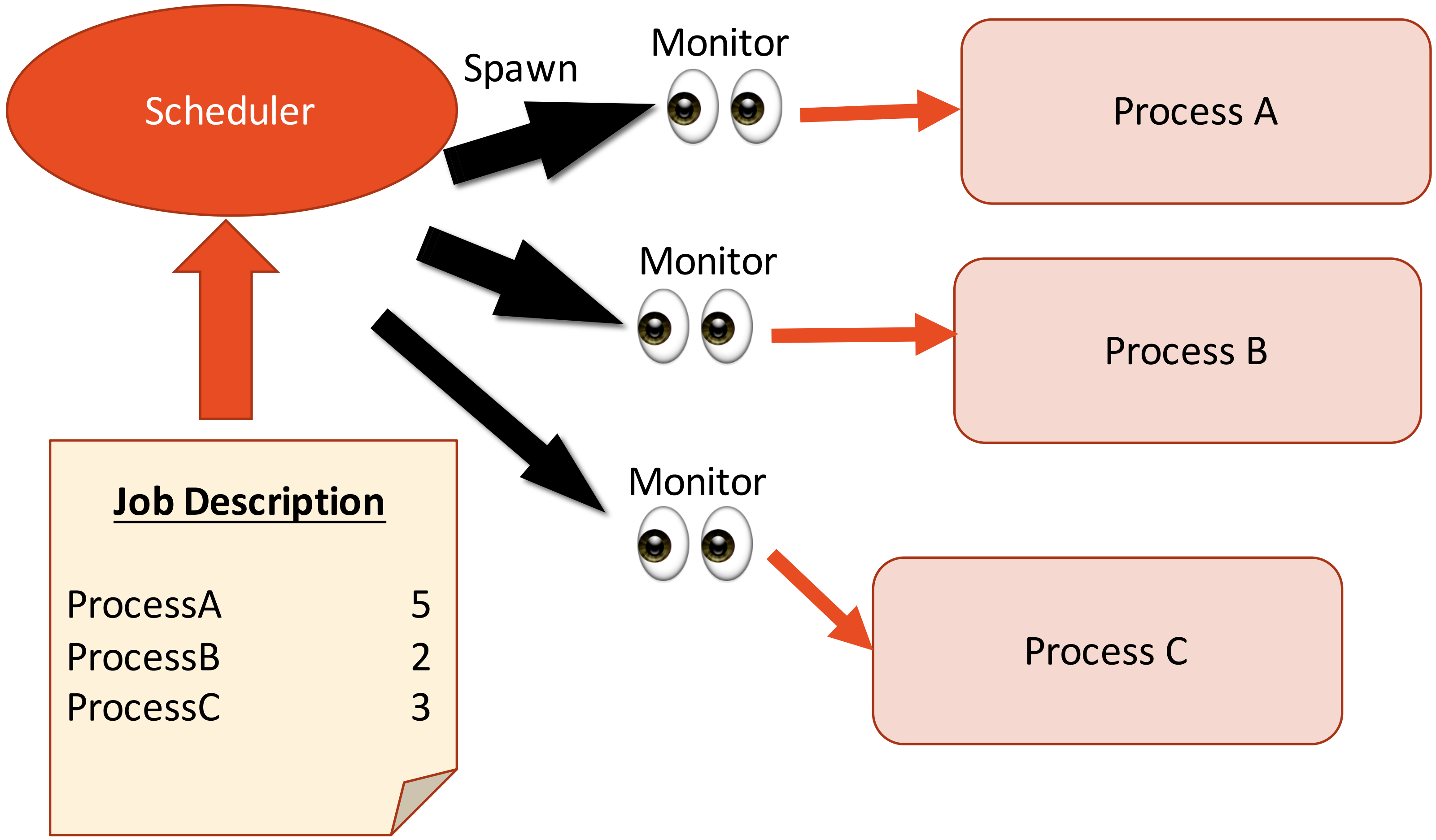
Job Description

ProcessA	5
ProcessB	2
ProcessC	3

Parallel (PARA Mode)

1. The scheduler reads the job description file (contains program name and duration).
2. It runs **ALL** jobs at the same time!!!
3. Scheduler counts the time:
    1. If the job ends **BEFORE** the scheduler, wake up the schedule.
    2. If time **EXCEEDS** the limit, the scheduler kills it.

Parallel (PARA Mode)



Job Description

ProcessA	5
ProcessB	2
ProcessC	3

Job Report

In this process the user no long needs to input jobs interactively.

The jobs are specified in a file.

Command	\t	Duration	\t
Command	\t	Duration	\t
Command	\t	Duration	\t

1. At most 10 jobs, and each line represents one job.
2. No built-in command!!
3. Assume all commands can be executed successfully.

Time

From the previous tutorial you know different types of time.

1. Time Elapsed
   - Time period from the start of the process till the termination.
2. CPU Time -User Time
   - Time spent on user code.
3. CPU Time - System Time
   - time spent on the kernel working on process's behalf.

Note that Time elapsed can be **NOT** equal to User Time + System Time!!!
- Usage of CPU, and etc….

We use real time instead of CPU time to count the process. Why?

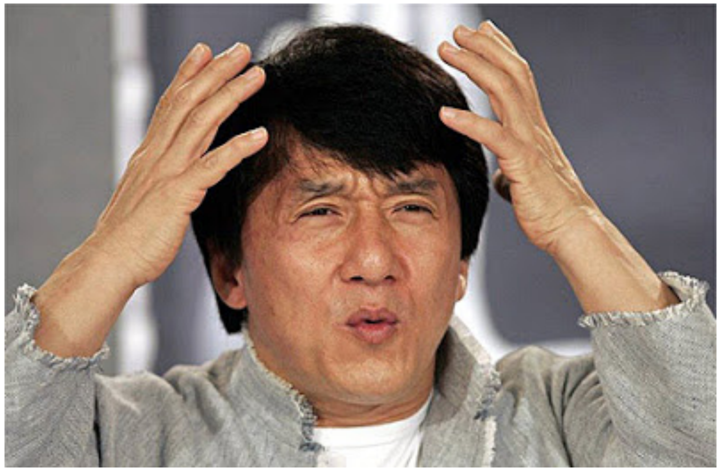
Chicken or the egg problem

1. To get the time, we need to wait until the process terminates.

2. However, we need a time to terminate the process in phase 2.

➔Chicken or the egg problem!!!!

Use of alarm() can do the job!!



How to get Time?

By using times(), you can get all the things you need ;)

```
#include <sys/times.h>
clock_t times(struct tms *buf);
```

CPU Time (User Time/System time): contains in tms buf structure.

Time Elapsed: Return Value - returns the **number of clock ticks** that have elapsed since an arbitrary point in the past.

BE AWARE!!
1.) You need to convert the time from clock ticks to seconds
2.) You need to have two elapsed time for calculating the time passed.

times()

```
#include <sys/times.h>
clock_t times(struct tms *buf);
```

```
#include <sys/times.h>
struct tms {
clock_t tms_utime; /* user time consumed */
clock_t tms_stime; /* system time consumed */
clock_t tms_cutime; /* user time consumed by children */
clock_t tms_cstime; /* system time consumed by children */
};
```

Get time of the process

```
#include <stdio.h> #include <sys/times.h> #include <unistd.h>
int main(int argc,char *argv[]){
        clock_t startTime,endTime;
        struct tms cpuTime;
        double ticks_per_sec = (double)sysconf(_SC_CLK_TCK);
        int i;
        startTime = times(&cpuTime);

        for(i = 0;i < 1000000000;i++);
        endTime = times(&cpuTime);
        printf("Time Elapsed: %.4f\n",(endTime-
startTime)/ticks_per_sec);
        printf("user time: %.4f\n",cpuTime.tms_utime/ticks_per_sec);
        printf("system time: %.4f\n",cpuTime.tms_stime/ticks_per_sec);
        return 0;
}
```

This is for current process, not children!

Get time of the process

```
$ ./1-time

Time Elapsed: 3.6900
user time: 3.6800
system time: 0.0000
```

1. It is normal 😉 that time elapsed and CPU time are different!!
2. Remember, convert the value to **seconds**!!
3. Use `tms_cutime, tms_cstime` for the children.

How to limit the running time

We can do the tricks by handling the SIGALRM signals!

What will we need to do?

1. Set a alarm of specified seconds.
2. When the alarm rings, call the handler.
3. In handler, kill the child if it does not finish.
4. If the child dies, wait() will unblock and we print out the report.

Signal Handler

```
pid_t pid;

void alrmHandler(int signal)
{
        kill(pid,SIGTERM);
}
```

It is simple for the handler, just send a SIGTERM signals to the child.

Limiting the running time

```
int main(int argc,char *argv[]) {
        signal(SIGALRM,alrmHandler);
        clock_t startTime,endTime;
        struct tms cpuTime;
        double ticks_per_sec = (double)sysconf(_SC_CLK_TCK);
        int i;
        startTime = times(&cpuTime);
        if(!(pid = fork())) {
                for(i = 0;i < 1000000000;i++);
                exit(0);
        }
        else{
                alarm(2);
                wait(NULL);
        }
        endTime = times(&cpuTime);
        // Printing Report
        return 0;
}
```

More troublesome: Parallel Mode

In parallel mode, you will need a middleman, a extra process to be the "monitor".

Reason:

➔ This can allow you to get the CPU time for each job.

Now a new difficulty arises:
How can we call the handler to kill a specific job?

Parallel Mode

You can set up an array for storing the monitor-job relationship.

Then when the SIGALRM has arrived, then you can search the array to find out the job pid.

Who is calling ?
- Monitor

```
pid_t pids[10][2];
void alrmHandler(int signal)
{
        pid_t myPid = getpid();
        // Searching for monitor pid
         kill(pids[i][1],SIGTERM);
         break;
}
```

Parallel Mode

```
for(j = 0;j < 3;j++){
// Fork 3 monitors
        if(!(pids[j][0] = fork())){
        /* Monitor Process*/
                signal(SIGALRM,alrmHandler);
                pids[j][0] = getpid();
                if(! (pids[j][1] = fork())){
                /* Job Process */
                        pids[j][1] = getpid();
                        // Doing Jobs
                }
                else{
                        alarm(SECOND);
                        wait(NULL);
                        // Print Time
                }
        }
        for (j = 0;j < 3;j++){waitpid(pids[j][0],NULL,0);}
}
```

Make Sure the array has changed

Near Final Destination!!! 😤

Deadline of Phase II :

23:59, 2016 Mar 18 (Fri)