Name: Sam Price

Student id: 7070819

Course: Computer Science

Module: 207SE Operating Systems, Security and Networks

Submission data: 23rd April

Portfolio 2

Table of Contents

Lab Activity 12 – Jobs 3

Lab Activity 13 – Job control 5

Lab Activity 14 – Linux command-line manipulation of processes 8

Lab Activity 15 IPC and Synchronisation 12

Lab Activity 17 TCP Server 14

Lab Activity 19 Security 15

Lab Activity 12 – Jobs

a) Comparing multiprogramming, multithreading and multitasking

Multiprogramming – "The overlapped or interleaved execution of two or more programs by a single CPU." (dictionary.com)

Figure 1 Multiprogramming tutorialspoint.com

Multiprogramming's main purpose is to maximise the use of CPU time. For example, if a process such as job 1 (in Figure 1) is performing an I/O task the OS may interrupt this to allow another in-main-memory programme, job 2, to proceed. This 'process context switching' helps prevent wasting CPU time in waiting for the I/O task to finish. The running process will continue to operate until it voluntarily releases the CPU or if it prevents an I/O operation from commencing. As long as there are processes to complete, the aim of keeping the CPU occupied is achieved ultimately by multiprogramming.

Although a hazard of multiprogramming is that one process may be modified by another. Therefore, the OS must have adequate protection and main memory must be able to hold multiple programmes in order to function effectively. Issues such as fragmentation can occur when processes enter and leave main memory.

Multitasking – When a CPU is able to "execute more than one program or task simultaneously" (oxforddictionaries.com)

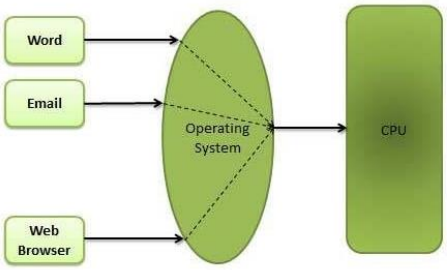


Figure 2 Multitasking tutorialspoint.com

When many processes, programs, tasks or threads are being run at the same time, they are being executed by the CPU by switching between them simultaneously. Tasks have to wait in line, while the CPU carries out one instruction at a time. Process context switching allows the CPU to reassign to another role giving the impression that multiple processes are working in parallel.

Multitasking and multiprogramming are both time sharing/saving systems, but they're subtly different. Multitasking refers to smaller units rather than entire processes. Multitasking is a more modern approach to time sharing, as it fairly distributes quantum (time) on the CPU rather than letting one program or job have access to it one at a time.

Multithreading – "a technique by which a single set of code can be used by several processors at different stages of execution" (oxforddictionaries.com)

Multiple code segments, or threads, can be part of a single process that can run alongside together within the system's context. This is Multithreading. Single threads work and execute separately, but they can rely on the main processes resources. Threads can run in parallel on a multiprocessing system or simply share a single CPU system.

Multithreading is a method of breaking down processes into smaller chunks so they can be executed concurrently on either a single CPU system or a multiprocessing system. Whereas multiprogramming and multitasking are purely time sharing systems.

What is a batch job?

A batch job is a sequence of commands listed in a file (batch file) to be executed by the operating system as a single unit (Indiana University, Knowledge Base). They are often accumulated during the working hours and executed over-night, for example bank transactions gathered throughout the day could be executed over night when no user input is required and/or when the CPU isn't busy. According to this article (helpsystems.com), batch jobs are essential to the modern enterprise. It has advantages such as lower costs (manual work is more expensive and expensive hardware isn't needed), maximising off the clock time (can be executed outside of business hours) and improving functions for businesses of all sizes (reduces time consuming tasks for businesses of all size).

Lab Activity 13 – Job control

a) Description of the activity

The program prints a menu displaying the following options;

1. Ls
2. Pwd
3. Ping www.apple.com
4. Ping www.sunderland.ac.uk and www.coventry.ac.uk

The system then outputs the command displaying the child's PID and the parents PID and its child PID.

b) Menu System

```
//fork and execl code to overcome problem of getting stuck in child process by using
wait()
#include <unistd.h>
#include <stdio.h>
#include "sys/types.h"
#include <sys/wait.h>
int main(){

pid_t pid;
int ping_fork;
//create a menu
printf("Please choose one of the following commands:\n1. ls\n2. pwd\n3 Ping
www.apple.com\n4 Ping www.coventry.ac.uk and www.sunderland.ac.uk\n");
int option;
//get the user input
scanf("%d",&option);
    int status = 0;
    int i;
    pid= fork() ;
        if(pid!=0) {
                printf ( " I am the parent my PID is %d, myPPID is %d, \n and my
child's PID is %d\n ",getpid(),getppid(),pid);
        }
       else {
              //To show the seperation of parent and child
              sleep(5);
              //use switch case to act on the user input
              switch (option){
                     case 1:
                            printf("I am the child my PID is %d.\n",getpid());
                            execl ("/bin/ls",".",(char*) NULL);
                            break;
                     case 2:
                            printf("I am the child my PID is %d.\n",getpid());
                            execl ("/bin/pwd",".",(char*) NULL);
                            break;
                     case 3:
                            printf("I am the child my PID is %d.\n",getpid());
                            execl ("/bin/ping", "ping", "-c 4","www.apple.com",
(char*) NULL);
                            break;
                     case 4:
                            if (pid == 0){
```

```
                                printf("I am the child my PID is %d.\n",getpid());
                                execl ("/bin/ping", "ping", "-c 4",
"www.sunderland.ac.uk", (char*) NULL);
                        }else {
                                ping_fork = fork();
                                if (ping_fork == 0){
                                        wait(&status);
                                        printf("I am the child my PID is
%d.\n",getpid());
                                        execl("/bin/ping", "ping","-c 4",
"www.coventry.ac.uk", (char*) NULL);
                                }
                        }
                        break;
                default:
                        printf("Invalid Selection");
                        break;
        }
    }
return 0;
}
```

The following output shows the results of options 1 to 3 on the menu.

```
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab13/fork_code$ ./lab_code
Please choose one of the following commands:
1. ls
2. pwd
3 Ping www.apple.com
4 Ping www.coventry.ac.uk and www.sunderland.ac.uk
1
 I am the parent my PID is 27189, myPPID is 9827,
 and my child's PID is 27191
 I am the child my PID is 27191.
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab13/fork_code$ for_fork.c  fork_execl.c  lab_code.c  s
imple_execl.c   twoproc_pid
fork_execl  lab_code      simple_execl  twoproc.c       twoproc_pid.c
./lab_code
Please choose one of the following commands:
1. ls
2. pwd
3 Ping www.apple.com
4 Ping www.coventry.ac.uk and www.sunderland.ac.uk
2
 I am the parent my PID is 27192, myPPID is 9827,
 and my child's PID is 27194
 I am the child my PID is 27194.
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab13/fork_code$ /home/207SE/prices25/SP207SE/Portfolio2
/lab13/fork_code
./lab_code
Please choose one of the following commands:
1. ls
2. pwd
3 Ping www.apple.com
4 Ping www.coventry.ac.uk and www.sunderland.ac.uk
3
 I am the parent my PID is 27195, myPPID is 9827,
 and my child's PID is 27196
 I am the child my PID is 27196.
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab13/fork_code$ PING e6858.dsce9.akamaiedge.net (104.68
.184.133) 56(84) bytes of data.
64 bytes from a104-68-184-133.deploy.static.akamaitechnologies.com (104.68.184.133): icmp_seq=1 ttl=
55 time=7.37 ms
64 bytes from a104-68-184-133.deploy.static.akamaitechnologies.com (104.68.184.133): icmp_seq=2 ttl=
55 time=7.08 ms
64 bytes from a104-68-184-133.deploy.static.akamaitechnologies.com (104.68.184.133): icmp_seq=3 ttl=
55 time=7.14 ms
64 bytes from a104-68-184-133.deploy.static.akamaitechnologies.com (104.68.184.133): icmp_seq=4 ttl=
55 time=7.43 ms

--- e6858.dsce9.akamaiedge.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 7.082/7.258/7.435/0.181 ms
```

The following screenshot shows the output of option 4.

```
Please choose one of the following commands:
1. ls
2. pwd
3 Ping www.apple.com
4 Ping www.coventry.ac.uk and www.sunderland.ac.uk
4
 I am the parent my PID is 27305, myPPID is 9827,
 and my child's PID is 27306
 I am the child my PID is 27306.
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab13/fork_code$ PING www-2017.sunderland.ac.uk (157.228.66.184)

--- www-2017.sunderland.ac.uk ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 2999ms
```

Lab Activity 14 – Linux command-line manipulation of processes

a) Process manipulation

- Example(s) of how to start processes.

Processes Started		
Process 1: $sleep 20	Process 2: $ping www.apple.com	Process 3: $yes

```
prices25@hvs-its-lnx01:~$ sleep 20
ls
```

Figure 3 Process 1

```
prices25@hvs-its-lnx01:~$ ping www.apple.com
PING e6858.dsce9.akamaiedge.net (23.63.86.162) 56(84) bytes of data.
64 bytes from a23-63-86-162.deploy.static.akamaitechnologies.com (23.63.86.162):
 icmp_seq=1 ttl=55 time=4.56 ms
```

Figure 4 Process 2

```
prices25@hvs-its-lnx01:~$ yes
y
y
y
y
y
y
y
```

Figure 5 Process 3

- Example(s) of how to suspend processes.

The Ctrl + z keys will suspend an application.	^Z
If the PID is known then the kill -stop command can be used to suspend an application.	Kill -STOP [pid]

The processes yes and sleep 20 were suspended using the ctrl + z comman

```
y
y
y
y
y
y^Z
[3]+  Stopped                 yes
```

```
prices25@hvs-its-lnx01:~$ sleep 20
ls
^Z
[1]+  Stopped                 sleep 20
```

The process sleep 100 was running in the background, the kill -STOP command was used to suspend it.

```
prices25@hvs-its-lnx01:~$ bg %4
[4]+ sleep 100 &
prices25@hvs-its-lnx01:~$ kill -STOP 17484
prices25@hvs-its-lnx01:~$ jobs
[3]-  Stopped                 yes
[4]+  Stopped                 sleep 100
```

- Example(s) of how to run process in background

A process can be run in the background by using the "$bg" command (if there is just one process being suspended), or by using the command "$bg %[job number]". To choose a specific job suspended at the time. Running the sleep 20 process in the background allows access to the command line as shown below in figure 6. You can also start a process running in the background using an "&" after the command (shown in figure 7).

```
prices25@hvs-its-lnx01:~$ jobs -l
[1]  16563 Stopped                 sleep 20
[2]- 16659 Stopped                 ping www.apple.com
[3]+ 16666 Stopped                 yes
prices25@hvs-its-lnx01:~$ bg %1
[1] sleep 20 &
prices25@hvs-its-lnx01:~$
```

Figure 6

Entering "$jobs" shows that the sleep 20 process was completed.

```
prices25@hvs-its-lnx01:~$ jobs
[1]   Done                    sleep 20
[2]-  Stopped                 ping www.apple.com
[3]+  Stopped                 yes
prices25@hvs-its-lnx01:~$
```

```
prices25@hvs-its-lnx01:~$ sleep 20 &
[5] 17970
prices25@hvs-its-lnx01:~$ jobs
[3]-  Stopped                 yes
[4]+  Stopped                 sleep 100
[5]   Running                 sleep 20 &
prices25@hvs-its-lnx01:~$
```

Figure 7

- Example(s) of how to run process in foreground and bring from background

A process can be run in the foreground from suspension or the background, using the "$fg" command (if there is just one process being suspended) or by using the command "$fg %[job number]". For example, sleep 30 is suspended and then resumed in the background and then brought forward to the foreground with the fg command.

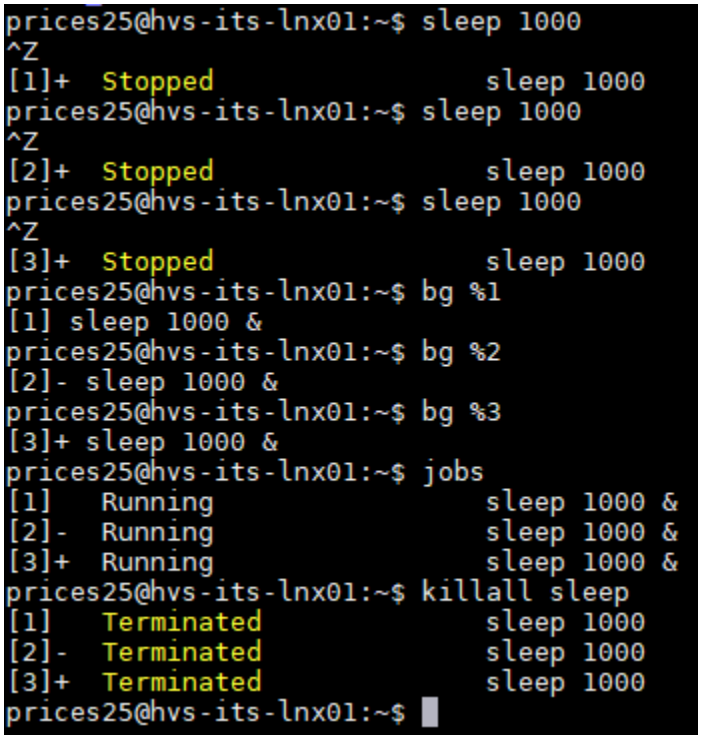
```
prices25@hvs-its-lnx01:~$ sleep 30
^Z
[1]+  Stopped                 sleep 30
prices25@hvs-its-lnx01:~$ bg %1
[1]+ sleep 30 &
prices25@hvs-its-lnx01:~$ jobs
[1]+  Running                 sleep 30 &
prices25@hvs-its-lnx01:~$ fg %1
sleep 30
prices25@hvs-its-lnx01:~$
```

- Example(s) of how to kill a process

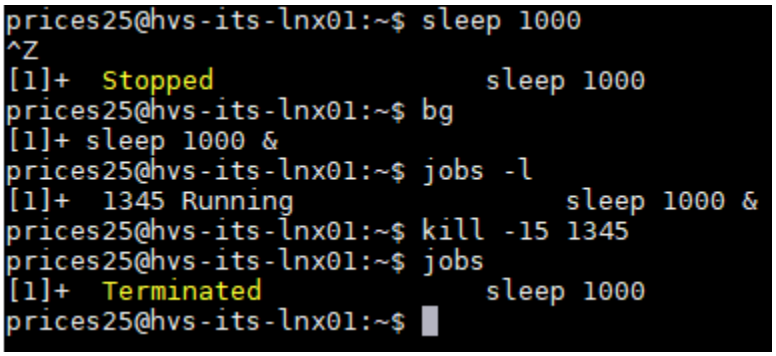
When using the kill command a specified signal can be sent to execute kill in different ways. You can either use the signal name (e.g. SIGTERM) or its signal number (e.g. 15). Pkill can be used to kill a process by using its name, however you must be careful to get its name correct. Killall can be used to kill all instances of a process.

- kill [signal name or number] [PID]
- pkill [process name]
- killall [signal name or number] process name

Example 1, killing all sleep processes using "killall sleep".



Example 2, safely killing a sleep process using SIGTERM (15) "kill -15 1345".



Example 3, terminate a process without saving SIGKILL (9) “kill -9

```
prices25@hvs-its-lnx01:~$ sleep 1000
^Z
[1]+  Stopped                 sleep 1000
prices25@hvs-its-lnx01:~$ bg
[1]+ sleep 1000 &
prices25@hvs-its-lnx01:~$ jobs -l
[1]+  1639 Running                 sleep 1000 &
prices25@hvs-its-lnx01:~$ kill -9 1639
prices25@hvs-its-lnx01:~$ jobs
[1]+  Killed                  sleep 1000
prices25@hvs-its-lnx01:~$
```

Example 4, kill a process using its process name “pkill sleep”.

```
prices25@hvs-its-lnx01:~$ sleep 1000
^Z
[1]+  Stopped                 sleep 1000
prices25@hvs-its-lnx01:~$ bg
[1]+ sleep 1000 &
prices25@hvs-its-lnx01:~$ pkill sleep
[1]+  Terminated              sleep 1000
prices25@hvs-its-lnx01:~$
```

b) Paragraph on disown, screen and nohup command

Screen, nohup and disown are commands that will continue to run a process once you have logged out or closed the SSH. Screen is a terminal multiplexer which allows it to access separate login sessions on a single terminal thus giving you the power to run a process and log out of the account. However, screen requires installation whereas nohup (no hang up) is built in. Disown allows you to detach a process running in the background and stops a SIGHUP signal being sent to the process once the terminal is closed so it continues to run.

c) Description and example of using watch command

The watch command allows you to run a program repeatedly every 2 seconds unless specified to do otherwise. The purpose is to allow you to see a change in output over time. The interval between which the process is run can be changed using -n [number].

watch free

```
Every 2.0s: free                                        Fri Apr 20 16:14:21 2018

              total        used        free      shared  buff/cache   available
Mem:       32938792      576160    31409752       29456      952880    31921332
Swap:       1046524           0     1046524
```

watch -n 5 free

```
Every 5.0s: free                                        Fri Apr 20 16:13:48 2018

              total        used        free      shared  buff/cache   available
Mem:       32938792      573408    31412584       29452      952800    31924116
Swap:       1046524           0     1046524
```

Lab Activity 15 IPC and Synchronisation

a) Brief description of activity

b) Modified semaphore example code so that the two processes output the song

The code can be found in Appendix B.

Screen shot of the code printing the song.

```
lab_code.c:160:9: note: each undeclared identifier is reported only once for each function it appears in
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab15/lab15$ nano lab_code.c
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab15/lab15$ gcc -o lab_code lab_code.c
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab15/lab15$ ./lab_code
Semaphore 87654 initialized.
In critical section P1 ...

 There's a hole in my bucket, dear Liza, dear Liza,
 There's a hole in my bucket, dear Liza, a hole.
Sleeping for 5 secs
Ending critical section P1 ...
In critical section P2 ...
 Then fix it, dear Henry, dear Henry, dear Henry,
 Then fix it, dear Henry, dear Henry, fix it.
Sleeping for 5 secs
Ending critical section P2 ...
In critical section P1 ...

 With what shall i fix it, dear Liza, dear Liza?
 With what shall i fix it, dear Liza, with what?
Sleeping for 3 secs
Ending critical section P1 ...
In critical section P2 ...
 With straw, dear Henry, dear Henry, dear Henry,
 With straw, dear Henry, dear Henry, with straw.
Sleeping for 3 secs
Ending critical section P2 ...
In critical section P1 ...

 The straw is too long, dear liza, dear Liza,
 The straw is too long, dear Liza, too long.
Sleeping for 5 secs
Ending critical section P1 ...
In critical section P2 ...
 Then cut it, dear Henry, dear Henry, dear Henry,
 Then cut it, dear Henry, dear Henry, cut it.
Sleeping for 5 secs
Ending critical section P2 ...
In critical section P1 ...

 With what shall i cut it, dear Liza, dear Liza?
 With what shall i cut it, dear Liza, with what?
Sleeping for 5 secs
Ending critical section P1 ...
In critical section P2 ...
 With an axe, dear Henry, dear Henry, dear Henry,
 With an axe, dear Henry, dear Henry, an axe.
Sleeping for 5 secs
Ending critical section P2 ...
In critical section P1 ...
prices25@hvs-its-lnx01:~/SP207SE/Portfolio2/lab15/lab15$
```

c) Modified code to write Liza part to stderr and redirect the two parts to different files.
[Changes to code above here with comments]
[Screenshot(s) showing code working]

Video Notes, how do processors communicate with each other

Deadlock – two processes prevent each other from accessing a resource

Live lock – processes involved in the live lock constantly change with regard to another, neither progressing.

Flag set by process 1, process 2 sleeps til flag is reset.

Issue is that operations that are testing and setting flags are separate.

Process 1	Process 2

... //wait for my turn
while (turn != 0); //busy wait
Critical Section
turn = 1;
...
	... //wait for my turn
	while (turn != 1); //busy wait
	Critical Section
	turn = 0;
	...

Satisifies mutual exclusion requirements, busy waitin and lockstep ynchronization

Lab Activity 17 TCP Server

a) Brief description of the TCP Server Activity

b) Commented Code showing changing to tcp-server.cc tcp-client.cc
[Commented code here]

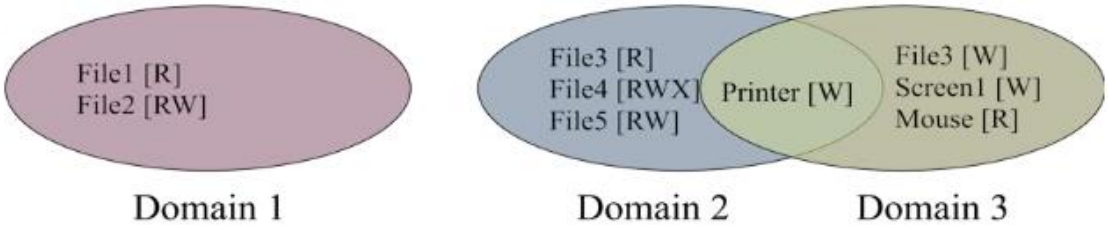
c) Example of TCP server parsing sentence for grammar correctness
[Screenshots showing server doing sentence parsing]

Or

d) Examples of TCP server doing booking system
[Screenshots showing server doing booking system]

Lab Activity 19 Security

a) Create a protection domain matrix, access list and capability list for the diagram below



Protection Domain Matrix								
	File 1	File 2	File 3	File 4	File 5	Printer	Screen	Mouse
D1	R	RW						
D2			R	RWX	RW	W		
D3			W				W	R

Access List

File 1	File 2	File 3
D1: <R>	D1: <R,W>	D2: <R> D3: <W>

File 4	File 5	Printer
D2: <R,W,X>	D2: <R,W>	D2: <W>

Screen 1	Mouse
D3: <W>	D3: <R>

Capability List

D1	D2	D3
File 1: <R> File 2: <R,W>	File 3: <R> File 4: <R,W,X> File 5: <R,W> Printer: <W>	File 3: <W> Screen: <W> Mouse: <R>

b) Commented Code showing unique hash function with salt

[Commented code here]

[outcomes of code here]

Salted Hash: Username + Salt + Password + Hash function

Appendix A

Appendix B

```
//Modified from
//http://docs.linux.cz/programming/c/unix_examples/semab.html

#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


#define KEY 87654 //Unique semaphore key



int main(){

   //int variables to track the number of times the semaphore has looped
   int P1Line = 1;
   int P2Line = 1;


  int id; /* Number by which the semaphore is known within a program */

  /* The next thing is an argument to the semctl() function. Semctl()
     does various things to the semaphore depending on which arguments
     are passed. We will use it to make sure that the value of the
     semaphore is initially 0. */

  union semun {
    int val;
    struct semid_ds *buf;
    ushort * array;
  } argument;

  argument.val = 1;

  /* Create the semaphore with external key KEY if it doesn't already
     exists. Give permissions to the world. */

  id = semget(KEY, 1, 0666 | IPC_CREAT);

  /* Always check system returns. */

  if(id < 0)
    {
      fprintf(stderr, "Unable to obtain semaphore.\n");
      exit(0);
    }

  /* What we actually get is an array of semaphores. The second
     argument to semget() was the array dimension - in our case
     1. */

  /* Set the value of the number 0 semaphore in semaphore array
     # id to the value 0. */

  if( semctl(id, 0, SETVAL, argument) < 0)
    {
      fprintf( stderr, "Cannot set semaphore value.\n");
```

```
    }
  else
    {
      fprintf(stderr, "Semaphore %d initialized.\n", KEY);
    }

  int pid=fork();

  if(pid){
    struct sembuf operations[1];
    int retval; /* Return value from semop() */

    /* Get the index for the semaphore with external name KEY. */
    id = semget(KEY, 1, 0666);

    if(id < 0){
      /* Semaphore does not exist. */

      fprintf(stderr, "Program sema cannot find semaphore, exiting.\n");
      exit(0);
    }
    operations[0].sem_num = 0;
    /* Which operation? Subtract 1 from semaphore value : */
    operations[0].sem_op = -1;
    /* Set the flag so we will wait : */
    operations[0].sem_flg = 0;

    while(1){
      //Process 1

      //wait
      operations[0].sem_op = -1;
      retval = semop(id, operations, 1);


      //critical section
      printf("In critical section P1 ... \n");
         //Using switch out put the lines for Liza depending on how many times the
process has been run
       switch(P1Line){
              case 1:
                     printf("\n There's a hole in my bucket, dear Liza, dear Liza,\n
There's a hole in my bucket, dear Liza, a hole.\n");
                     break;
              case 2:
                     printf("\n With what shall i fix it, dear Liza, dear Liza?\n With
what shall i fix it, dear Liza, with what?\n");
                     break;
              case 3:
                     printf("\n The straw is too long, dear liza, dear Liza,\n The
straw is too long, dear Liza, too long.\n");
                     break;
              case 4:
                     printf("\n With what shall i cut it, dear Liza, dear Liza?\n With
what shall i cut it, dear Liza, with what?\n");
                     break;
              case 5:
                     //exit once the song has finished.
                     exit(0);
                     break;
              default:
                     printf("\n Songs finished.\n");
```

```
                    break;
}
       P1Line++;
      fflush(stdout);
      int stime=2+(rand()/(float)(RAND_MAX))*4;
      printf("Sleeping for %d secs\n",stime);
      sleep(stime);


      printf("Ending critical section P1 ... \n");
      fflush(stdout);

      operations[0].sem_op = 1;
      //signal
      retval = semop(id, operations, 1);

    }
  }else{
    //Process 2
    struct sembuf operations[1];
    int retval; /* Return value from semop() */
    /* Get the index for the semaphore with external name KEY. */
    id = semget(KEY, 1, 0666);
    if(id < 0){
      /* Semaphore does not exist. */

      fprintf(stderr, "Program sema cannot find semaphore, exiting.\n");
      exit(0);
    }
    operations[0].sem_num = 0;
    /* Which operation? Subtract 1 from semaphore value : */
    operations[0].sem_op = -1;
    /* Set the flag so we will wait : */
    operations[0].sem_flg = 0;

    while(1){



      //wait
      operations[0].sem_op = -1;
      retval = semop(id, operations, 1);

      //critical section
      printf("In critical section P2 ... \n");
         //Using switch out put the lines for Liza depending on how many times the
process has been run
       switch(P2Line){
              case 1:
                    printf(" Then fix it, dear Henry, dear Henry, dear Henry,\n Then
fix it, dear Henry, dear Henry, fix it.\n");
                    break;
              case 2:
                    printf(" With straw, dear Henry, dear Henry, dear Henry,\n With
straw, dear Henry, dear Henry, with straw.\n");
                    break;
              case 3:
                    printf(" Then cut it, dear Henry, dear Henry, dear Henry,\n Then
cut it, dear Henry, dear Henry, cut it.\n");
                    break;
              case 4:
```

```
                    printf(" With an axe, dear Henry, dear Henry, dear Henry,\n With
an axe, dear Henry, dear Henry, an axe.\n");
                    break;
             case 5:
                    //exit once the song has finished.
                    exit(0);
                    break;
             default:
                    printf("\n Songs finished.\n");
                    break;
}
       P2Line++;
      fflush(stdout);
      int stime=2+(rand()/(float)(RAND_MAX))*4;
      printf("Sleeping for %d secs\n",stime);
      sleep(stime);

      printf("Ending critical section P2 ... \n");
      fflush(stdout);

      //signal
      operations[0].sem_op = 1;
      retval = semop(id, operations, 1);

    }

  }

}
```

References

17/4/2018

https://kb.iu.edu/d/afrx

https://www.helpsystems.com/resources/articles/batch-processing-how-it-evolved-and-how-your-business-benefits

http://www.dictionary.com/browse/multiprogramming

https://en.oxforddictionaries.com/definition/multitask

https://en.oxforddictionaries.com/definition/multithreading

https://www.tutorialspoint.com/operating_system/os_properties.htm

http://www.8bitavenue.com/2012/10/difference-between-multiprogramming-multitasking-multithreading-and-multiprocessing/

20/4/18

https://stephenbridgett.wordpress.com/2013/04/15/screen-nohup-and-disown/

http://www.linfo.org/watch.html