Student ID: 8971405

Course: Ethical Hacking & Cybersecurity

Module: 207se Operating Systems, Security & Networks

COVENTRY UNIVERSITY

PORTFOLIO CONTENTS

LAB ACTIVITY 12 – MULTIPROCESSING

A. COMPARISON OF THE MULTITASKING, MULTIPROGRAMMING AND MULTITHREADING.

MULTITASKING

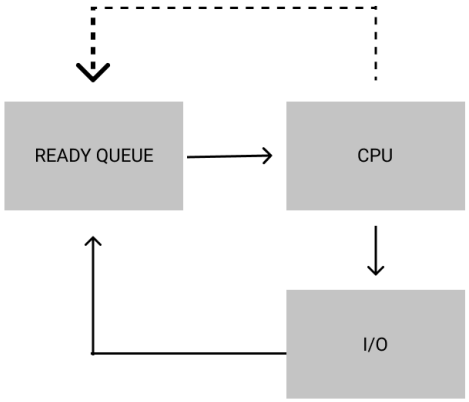
Conventional wisdom suggests multitasking to be multiple tasks being run simultaneously, however (Abualrob 2012) suggests multitasking to be the illusion of parallelism by utilising context switching to assign different tasks to the CPU at any given time.

Multitasking can exist on a process level and also a thread level known as multithreading (discussed later). The key point is that any task (thread or process) is sharing common processing resources like memory or the CPU which are reallocated accordingly as tasks are switched between one to another.

MULTIPROGRAMMING

Take for instance queueing up to order food at MCD. When it is my turn to order and I have not decided whether I would like a cheeseburger or a big mac, I would stand to the side while the customer behind me can proceed to order while I make up my mind.

This is similar to what multiprogramming does. When it is a process' turn to be moved into the CPU from the ready-state queue but needs additional resources eg. I/O request/sub process, it will be sent to another part of memory thus freeing up resources for other processes to run until the I/O request has been fulfilled, thereby maximising the utilization of resources. After which the process can be moved back into the ready-state queue for execution.



MULTITHREADING

Multithreading as discussed earlier, allows processes to have different segments (threads) that run concurrently in the same process. (Tolomei 2017) puts it in understandable way stating multithreading to be similar to a parent process having multiple child processes that run independently, all sharing the same resources of the parent process. Multithreading, like multiprogramming and multitasking, greatly increases the efficiency of process execution and allows for improved throughput of computers.

EVOLUTION OVERTIME & SUMMARY

The high costs of CPU execution time during the early days of computing meant execution of processes needed to be optimised to max out the usage of the CPU. This led to the creation of multiprogramming.

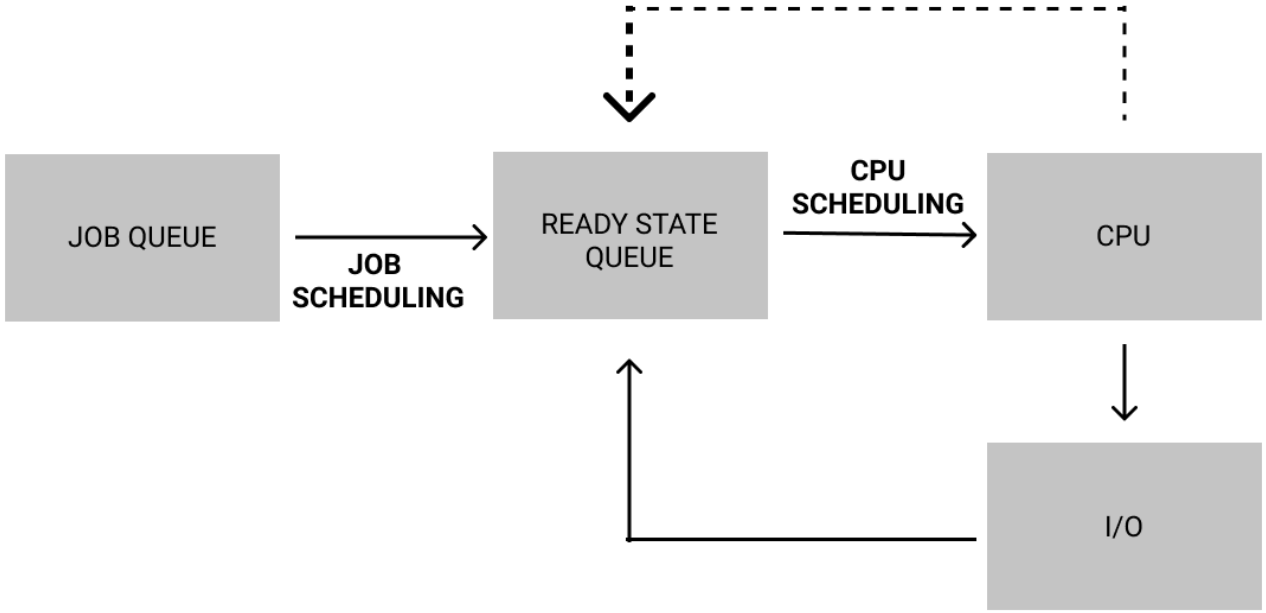
Similarly, multitasking employs the use of context switching which swap processes in and out of memory to make full use of computer hardware. The difference between multitasking and multiprogramming is that the latter is a subset of the former and that multitasking is used more in modern OSes.

Multithreading was created with the aim of optimising the limitation (back then) of a single core by creating multiple threads that share resources with a parent process.

All of these methods improve the efficiency of computers execution. They stem from the same economical problem that was present during the beginnings of computing and vastly helped with driving down the need for more hardware to obtain speed.

B. COMPARISON OF CPU AND JOB SCHEDULING.

Scheduling is the allocation of resources to tasks in a computer. There exist 2 types of scheduling namely CPU Scheduling and Job Scheduling. The latter is responsible for task selection that will be brought from the job queue to the ready-state queue (secondary memory to main memory). From there, CPU scheduling steps in and utilises an algorithm to select which processes in the ready-state queue will be executed by the CPU next (how processes in main memory get CPU execution time). The following diagram illustrates this in more detail.



LAB ACTIVITY 13 – JOB CONTROL

A. DESCRIPTION OF THE ACTIVITY

This activity is about multiprocessing by using the fork() function to create a parent & child process each with its own unique PID that can be viewed with the ps command.

This activity takes a look at creating a menu system where a user can choose which command they want to run, and the program will fork the process into 2 processes- the child and the parent. Depending on the command, the parent will have to wait until the child process is finished before it can execute its portion of code.

This activity extends to explore the uses of signals, kill() and pause() by incorporating them into the menu system.

B. MENU SYSTEM

[Commented menu fork code]

```
//Menu system that uses the fork() function to run multiple processes by creating parent
and a child process.
#include <unistd.h>
#include <stdio.h>
#include "sys/types.h"
#include <sys/wait.h>
int main()
{
    int a = 0;
    for (a = 0; a <= 5; a++)
    {
        //creating all the variables used
        int input;
        int input2;
        pid_t pid_value;
        int status = 0;
        //prints out a menu option to allow users to choose options 1-3
        //takes user input and stores it in variable int input;
        printf("Please select option 1 - 6\n");
        printf("1. ps\n");
        printf("2. date\n");
        printf("3. ifconfig\n");
        printf("4. ls\n");
        printf("5. ping\n");
        printf("6. pause\n");
        printf("Enter your option: ");
        scanf("%d", &input);
        //checks input if input is 1-6
        if (input == 1)
        {
            pid_value = fork();
            if (pid_value != 0)
            {
                wait(&status);
```

```
                //since the return value of the fork() fn is the child's pid, we can use t
hat to show the value of the child's pid
                printf("====================================\n ");
                printf("I am the parent my Process ID is %d, myChild's PID is %d, \n ",
                getpid(), pid_value);
                printf("Using the wait ensures that my child finishes first. \n ");
            }
            else
            {
                printf("I am the child, my Process ID is %d , my Parents PID is %d \n",
                getpid(), getppid());
                sleep(2);
                printf("Using execl to display running processes\n");
                printf("====================================\n ");
                execl("/bin/ps", "ps", (char *)0);
            }
        }
        //Date command
        else if (input == 2)
        {
            pid_value = fork();
            if (pid_value != 0)
            {
                wait(&status);
                printf("====================================\n ");
                printf("I am the parent my Process ID is %d, myChild's PID is %d, \n ",
                getpid(), pid_value);
                printf("Using the wait ensures that my child finishes first. \n ");
            }
            else
            {
                printf("I am the child, my Process ID is %d , my Parents PID is %d \n",
                getpid(), getppid());
                sleep(2);
                printf("Using execl to display the current date & time.\n");
                printf("====================================\n ");
                execl("/bin/date", "date", (char *)0);
            }
        }
        //ifconfig command
        else if (input == 3)
        {
            pid_value = fork();
            if (pid_value != 0)
            {
                wait(&status);
                printf("====================================\n ");
                printf("I am the parent my Process ID is %d, myChild's PID is %d, \n ",
                getpid(), pid_value);
                printf("Using the wait ensures that my child finishes first. \n ");
            }
```

```
            else
            {
                printf("I am the child, my Process ID is %d , my Parents PID is %d \n", ge
tpid(), getppid());
                sleep(2);
                printf("Lisitng all network interfaces.\n");
                printf("====================================\n ");
                execl("/sbin/ifconfig", "ifconfig", (char *)0);
            }
        }
        //ls command
        else if (input == 4)
        {
            pid_value = fork();
            int child_pid;
            if (pid_value != 0)
            {
                wait(&status);
                printf("====================================\n ");
                printf("I am the parent my Process ID is %d, myChild's PID is %d, \n ",
                getpid(), pid_value);
                printf("Using the wait ensures that my child finishes first.\n ");
            }
            else
            {
                printf("I am the child, my Process ID is %d , my Parents PID is %d \n",
                getpid(), getppid());
                sleep(2);
                printf("Listing files in current directory\n");
                printf("====================================\n ");
                execl("/bin/ls", "ls", (char *)0);
            }
        }
        //ping command for Google and Yahoo
        else if (input == 5)
        {
            pid_value = fork();
            int child_pid;
            if (pid_value != 0)
            {
                wait(&status);
                printf("------------------------------------------------------------------
--------\n ");
                printf(" I am the parent my Process ID is %d, myChild's PID is %d, \n ",
                getpid(), pid_value);
                printf("I am pinging yahoo.com with 2 packets\n ");
                printf("====================================\n ");
                execl("/bin/ping", "ping","www.yahoo.com","-c","2" ,(char *)0);
                //the parent pings yahoo.com 2 packets
            }
            else
```

```
        {
            printf("I am the child, my Process ID is %d , my Parents PID is %d \n",
            getpid(), getppid());
            sleep(4);
            printf("I am pinging www.google.com with 2 packets\n");
            printf("====================================\n ");
            execl("/bin/ping", "ping","www.google.com","-c","2" ,(char *)0);
            //the child pings google.com 2 packets
        }
    }
    else if (input == 6)
    {
        pid_value = fork();
        int child_pid;
        if(pid_value != 0)
        {
            printf("I am the parent my Process ID is %d, myChild's PID is %d, \n ",
            getpid(), pid_value);
            printf("Pausing child process...\n");
            kill(pid_value, SIGSTOP);
            execl("/bin/ps", "ps", (char *)0);
        }
        else
        {
            printf("I am the child, my Process ID is %d , my Parents PID is %d \n",
            getpid(), getppid());
            sleep(2);
            execl("/bin/ping", "ping","www.google.com", (char *)0);
            //The child doesn't actually ping as the parent pauses it.
        }
    }
    //if input is not 1 - 6, it's invalid input.
    else
    {
        printf("Invalid Input\n");
    }
    return 0;
}
```

[Output from code]

Option [1]: ps

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$ ./fork_execl
Please select option 1 - 6
1. ps
2. date
3. ifconfig
4. ls
5. ping
6. pause
Enter your option: 1
I am the child, my Process ID is 10913 , my Parents PID is 10912
Using execl to display running processes
====================================
  PID TTY          TIME CMD
10725 pts/0    00:00:00 bash
10912 pts/0    00:00:00 fork_execl
10913 pts/0    00:00:00 ps
====================================
 I am the parent my Process ID is 10912, myChild's PID is 10913,
 Using the wait ensures that my child finishes first.
 shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

//As shown, the child process prints its PID as well as its parent's PID while the parent prints its own PID as well as its child's PID.

Option [2]: date

```
Enter your option: 2
I am the child, my Process ID is 10969 , my Parents PID is 10968
Using execl to display the current date & time.
====================================
Sat 11 Apr 07:21:45 BST 2020
====================================
 I am the parent my Process ID is 10968, myChild's PID is 10969,
 Using the wait ensures that my child finishes first.
 shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

Option [3]: ifconfig

```
Enter your option: 3
I am the child, my Process ID is 11003 , my Parents PID is 11002
Lisitng all network interfaces.
====================================
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.50.80.73  netmask 255.255.252.0  broadcast 10.50.83.255
        inet6 fe80::21d:d8ff:feb7:1d78  prefixlen 64  scopeid 0x20<link>
        ether 00:1d:d8:b7:1d:78  txqueuelen 1000  (Ethernet)
        RX packets 89922  bytes 16069852 (16.0 MB)
        RX errors 0  dropped 11534  overruns 0  frame 0
        TX packets 21398  bytes 23003247 (23.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 166  bytes 13274 (13.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 166  bytes 13274 (13.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

====================================
 I am the parent my Process ID is 11002, myChild's PID is 11003,
 Using the wait ensures that my child finishes first.
 shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

Option [4]: ls

```
Enter your option: 4
I am the child, my Process ID is 11037 , my Parents PID is 11036
Listing files in current directory
====================================
for_execl.c  fork  fork.c  fork_execl  fork_execl.c.save  simple_execl.c  twoproc.c  twoproc_pid.c
====================================
 I am the parent my Process ID is 11036, myChild's PID is 11037,
 Using the wait ensures that my child finishes first.
 shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

Option [5]: ping

```
Enter your option: 5
I am the child, my Process ID is 11039 , my Parents PID is 11038
I am pinging www.google.com with 2 packets
==================================
PING www.google.com (172.217.169.4) 56(84) bytes of data.
64 bytes from 172.217.169.4 (172.217.169.4): icmp_seq=1 ttl=48 time=10.3 ms
64 bytes from 172.217.169.4 (172.217.169.4): icmp_seq=2 ttl=48 time=10.4 ms

--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 10.370/10.406/10.442/0.036 ms
--------------------------------------------------------------------------------
  I am the parent my Process ID is 11038, myChild's PID is 11039,
 I am pinging yahoo.com with 2 packets
 ==================================
PING atsv2-fp-shed.wg1.b.yahoo.com (87.248.98.8) 56(84) bytes of data.
64 bytes from media-router-fp2.prod1.media.vip.ir2.yahoo.com (87.248.98.8): icmp_seq=1 ttl=47 time=22.6 ms
64 bytes from media-router-fp2.prod1.media.vip.ir2.yahoo.com (87.248.98.8): icmp_seq=2 ttl=47 time=22.5 ms

--- atsv2-fp-shed.wg1.b.yahoo.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 22.578/22.607/22.637/0.153 ms
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

Option [6]: pause child process

```
Enter your option: 6
I am the parent my Process ID is 11081, myChild's PID is 11082,
 Pausing child process...
  PID TTY          TIME CMD
10725 pts/0    00:00:00 bash
11081 pts/0    00:00:00 ps
11082 pts/0    00:00:00 fork_execl
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session13/fork_code$
```

//As shown, the parent process pauses the child process by sending a SIGSTOP signal. It then executes ps to show that the child process (PID: 11082) is still alive and can be restarted sending a SIGCONT signal if needed.

LAB ACTIVITY 15 IPC AND SYNCHRONISATION

A. BRIEF DESCRIPTION OF ACTIVITY

This activity is about process synchronisation by utilizing semaphores. The task involves creating a script that prints out lyrics from a song “A hole in my bucket” where 2 characters will take turns to sing their lines which uses the fork() function from lab 13.

Typically processes that run at the same time will encounter the problem of collision where both processes choose to run at the same time and causes an error. The use of semaphores allows both processes to be run in sync while not encountering any collision. This allows the lyrics to be printed back and forth (in this case by Henry and Liza) by each process until all the lyrics have been successfully read to stdout.

B. MODIFIED SEMAPHORE EXAMPLE CODE SO THAT THE TWO PROCESSES OUTPUT THE SONG.

[Comment code here]

```
//This is script uses semaphores for processes to take turns in executions to prevent
//2 or more processes executing at the same time.
//This example prints the lyrics for the song "A hole in my bucket" where 2 processes take
s turns to print their line of the song.
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "ops_sems.h"
int main(int argc, char argv[])
{
  int id;
  //Use our source file as the "key"
  id = ops_semget("critical_example2.c", 1);
  int pid = fork();
  if (pid)
  {
    //P1
    while (1)
    {
      //All of Henry's lines are grouped together in process 1
      ops_wait(id);
      printf("There's a hole in the bucket, dear Liza, dear Liza,\n");
      printf("There's a hole in the bucket, dear Liza, a hole.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      printf("With what shall I fix it, dear Liza, dear Liza?\n");
      printf("With what shall I fix it, dear Liza, with what?\n");
```

```
      rsleep();

      ops_signal(id);
      //Wrong lyrics spotted, corrected (extra points?)
      ops_wait(id);
      printf("The stick is too big, dear Liza, dear Liza,\n");
      printf("The stick is too big, dear Liza, too big.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      printf("With what shall I cut it, dear Liza, dear Liza?\n");
      printf("With what shall I cut it, dear Liza, with what?\n");
      rsleep();
      ops_signal(id);
    }
  }
  else
  {
    //Liza's lines are group together under process 2
    while (1)
    {
      ops_wait(id);
      printf("Then fix it, dear Henry, dear Henry, dear Henry,\n");
      printf("Then fix it, dear Henry, dear Henry, fix it.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      printf("With a stick, dear Henry, dear Henry, dear Henry,\n");
      printf("With a stick, dear Henry, dear Henry, a stick.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      printf("Then cut it, dear Henry, dear Henry, dear Henry,\n");
      printf("Then cut it, dear Henry, dear Henry, cut it.\n");
      rsleep();
      ops_signal(id);

      //Wrong lyrics spotted, corrected (extra points?)
      ops_wait(id);
      printf("With a hatchet, dear Henry, dear Henry, dear Henry,\n");
      printf("With a hatchet, dear Henry, dear Henry, a hatchet.\n");
      rsleep();
      ops_signal(id);
    }
  }
}
```

[Screenshot(s) showing code working]

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$ ./song
Semaphore 17960336 initialized with path 'critical_example2.c'.
There's a hole in the bucket, dear Liza, dear Liza,
There's a hole in the bucket, dear Liza, a hole.

Then fix it, dear Henry, dear Henry, dear Henry,
Then fix it, dear Henry, dear Henry, fix it.

With what shall I fix it, dear Liza, dear Liza?
With what shall I fix it, dear Liza, with what?

With a stick, dear Henry, dear Henry, dear Henry,
With a stick, dear Henry, dear Henry, a stick.

The stick is too big, dear Liza, dear Liza,
The stick is too big, dear Liza, too big.

Then cut it, dear Henry, dear Henry, dear Henry,
Then cut it, dear Henry, dear Henry, cut it.

With what shall I cut it, dear Liza, dear Liza?
With what shall I cut it, dear Liza, with what?

With a hatchet, dear Henry, dear Henry, dear Henry,
With a hatchet, dear Henry, dear Henry, a hatchet.
```

C. MODIFIED CODE TO WRITE LIZA'S PART TO STDERR AND REDIRECT THE TWO PARTS TO A FILE.

[Changes to code above here with comments]

```c
//This is script uses the semaphores for processes to take turns in executions to prevent
//2 or more processes executing at the same time.
//This example prints the lyrics for the song "A hole in my bucket" where 2 processes take
s
//turns to print their line of the song.
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "ops_sems.h"
int main(int argc, char argv[])
{
  int id;
  //Use our source file as the "key"
  id = ops_semget("critical_example2.c", 1);
  int pid = fork();
  FILE *henry;  //Creates 2 file pointers for henry and liza to store the lyrics.
  FILE *liza;

  if (pid)
  {
    //P1
    henry = fopen("henry.txt", "w"); //Opens a file “henry.txt” in write mode.
    int count_henry = 0; //Count for henry so it only goes through one loop
    while(count_henry<1)
    {
```

```
      //All of Henry's lines are groupped together in process 1
      ops_wait(id);
      //Using fprintf writes the text to the file.
      fprintf(henry,"There's a hole in the bucket, dear Liza, dear Liza,\n");
      fprintf(henry,"There's a hole in the bucket, dear Liza, a hole.\n");
      //Printing to the screen
      printf("There's a hole in the bucket, dear Liza, dear Liza,\n");
      printf("There's a hole in the bucket, dear Liza, a hole.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      fprintf(henry,"With what shall I fix it, dear Liza, dear Liza?\n");
      fprintf(henry,"With what shall I fix it, dear Liza, with what?\n");
      //Printing to the screen
      printf("With what shall I fix it, dear Liza, dear Liza?\n");
      printf("With what shall I fix it, dear Liza, with what?\n");
      rsleep();

      ops_signal(id);
      //Wrong lyrics spotted, corrected (extra points?)
      ops_wait(id);
      fprintf(henry,"The stick is too big, dear Liza, dear Liza,\n");
      fprintf(henry,"The stick is too big, dear Liza, too big.\n");
      //Printing to the screen
      printf("The stick is too big, dear Liza, dear Liza,\n");
      printf("The stick is too big, dear Liza, too big.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      fprintf(henry,"With what shall I cut it, dear Liza, dear Liza?\n");
      fprintf(henry,"With what shall I cut it, dear Liza, with what?\n");
      //Printing to the screen
      printf("With what shall I cut it, dear Liza, dear Liza?\n");
      printf("With what shall I cut it, dear Liza, with what?\n");

      rsleep();
      ops_signal(id);
      count_henry++; //Increase count by 1 to loop only once
    }
    fclose(henry);  //Closes the file to free up memory.
  }
  else
  {
    //Liza's lines are group together under process 2
    liza = freopen("liza.txt", "w" , stderr); //Liza's part is written to stderr
    int count_liza = 0; //Count for liza so it only goes through one loop
    while(count_liza<1)
    {
      ops_wait(id);
```

```
      fprintf(liza,"Then fix it, dear Henry, dear Henry, dear Henry,\n");
      fprintf(liza,"Then fix it, dear Henry, dear Henry, fix it.\n");
      //Printing to the screen
      printf("Then fix it, dear Henry, dear Henry, dear Henry,\n");
      printf("Then fix it, dear Henry, dear Henry, fix it.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      fprintf(liza,"With a stick, dear Henry, dear Henry, dear Henry,\n");
      fprintf(liza,"With a stick, dear Henry, dear Henry, a stick.\n");
      //Printing to the screen
      printf("With a stick, dear Henry, dear Henry, dear Henry,\n");
      printf("With a stick, dear Henry, dear Henry, a stick.\n");
      rsleep();
      ops_signal(id);

      ops_wait(id);
      fprintf(liza,"Then cut it, dear Henry, dear Henry, dear Henry,\n");
      fprintf(liza,"Then cut it, dear Henry, dear Henry, cut it.\n");
      //Printing to the screen
      printf("Then cut it, dear Henry, dear Henry, dear Henry,\n");
      printf("Then cut it, dear Henry, dear Henry, cut it.\n");
      rsleep();
      ops_signal(id);

      //Wrong lyrics spotted, corrected (extra points?)
      ops_wait(id);
      fprintf(liza,"With a hatchet, dear Henry, dear Henry, dear Henry,\n");
      fprintf(liza,"With a hatchet, dear Henry, dear Henry, a hatchet.\n");
      //Printing to the screen
      printf("With a hatchet, dear Henry, dear Henry, dear Henry,\n");
      printf("With a hatchet, dear Henry, dear Henry, a hatchet.\n");
      rsleep();
      ops_signal(id);
      count_liza++; //Increase count by 1 to loop only once
    }
    fclose(liza);  //Closes the file to free up memory.
  }
}
```

As shown, the above code prints the song to the screen as well as write the lyrics to individual files labelled henry.txt and liza.txt

This was achieved by writing the lyrics twice, however, I believe it is possible to achieve the same result without duplication of data. Probably for future research.

[Screenshot(s) showing code working]

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$ ./song
Semaphore 17960336 initialized with path 'critical_example2.c'.
There's a hole in the bucket, dear Liza, dear Liza,
There's a hole in the bucket, dear Liza, a hole.

Then fix it, dear Henry, dear Henry, dear Henry,
Then fix it, dear Henry, dear Henry, fix it.

With what shall I fix it, dear Liza, dear Liza?
With what shall I fix it, dear Liza, with what?

With a stick, dear Henry, dear Henry, dear Henry,
With a stick, dear Henry, dear Henry, a stick.

The stick is too big, dear Liza, dear Liza,
The stick is too big, dear Liza, too big.

Then cut it, dear Henry, dear Henry, dear Henry,
Then cut it, dear Henry, dear Henry, cut it.

With what shall I cut it, dear Liza, dear Liza?
With what shall I cut it, dear Liza, with what?

With a hatchet, dear Henry, dear Henry, dear Henry,
With a hatchet, dear Henry, dear Henry, a hatchet.

shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$
```

[Screenshot henry.txt]

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$ cat henry.txt
There's a hole in the bucket, dear Liza, dear Liza,
There's a hole in the bucket, dear Liza, a hole.
With what shall I fix it, dear Liza, dear Liza?
With what shall I fix it, dear Liza, with what?
The stick is too big, dear Liza, dear Liza,
The stick is too big, dear Liza, too big.
With what shall I cut it, dear Liza, dear Liza?
With what shall I cut it, dear Liza, with what?
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$
```

[Screenshot of liza.txt]

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$ cat liza.txt
Then fix it, dear Henry, dear Henry, dear Henry,
Then fix it, dear Henry, dear Henry, fix it.
With a stick, dear Henry, dear Henry, dear Henry,
With a stick, dear Henry, dear Henry, a stick.
Then cut it, dear Henry, dear Henry, dear Henry,
Then cut it, dear Henry, dear Henry, cut it.
With a hatchet, dear Henry, dear Henry, dear Henry,
With a hatchet, dear Henry, dear Henry, a hatchet.
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session15/lab15$
```

LAB ACTIVITY 16 CONSUMER/PRODUCER

A. BRIEF DESCRIPTION OF CONSUMER/PRODUCER

The consumer/ producer problem is a common example of the synchronisation problem. According to (Zhang et al. 2009) both the produce & consumer share a common fixed-length buffer. The producer is responsible for data insertion while the consumer consumes data simultaneously. The problem is to ensure that both processes do not clash i.e. The producer does not add more data to a filled buffer and the consumer does not pop data from an empty buffer.

B. MODIFIED CODE TO SOLVE CONSUMER/PRODUCER PROBLEM. SHOWING DIFFERENT BUFFER LENGTHS AND DIFFERENT SPEEDS.

[Comment code here]

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "ops_sems.h"
#include <sys/wait.h>
/* Remember to try reversing the timings...*/
int bufferlength = 8; //Limited buffer length
//what could we do about this?
int main(int argc, char argv[])
{
  //Created 2 variables to store the positions of the consumer & producer relative to the buffer length.
  int conslot = bufferlength + 1;
  int prodslot = bufferlength + 2;
  pid_t pid;
  int status = 0;
  int i;
  //Create shared memory segment
  int shm_id = shmget(ftok("prodcon_example2.c", 2), bufferlength,
                      0666 | IPC_CREAT);
  //Use our source file as the "key"
  int id = ops_semget("prodcon_example2.c", 0);
  char *data; //For our pointer to shared memory...
  pid = fork();
  if (pid)
  {
    //P1 - CONSUMER
    shm_id = shmget(ftok("prodcon_example2.c", 2), 0, 006);
    //Attach the shared buffer
    data = shmat(shm_id, (void *)0, 0);
    //Initialise the positions of the consumer & producer to 0
    data[conslot] = 0;
    data[prodslot] = 0;
```

```
    while (1){ //Runs the code indefinitely
      //If the position of the producer is equaled to the position of the consumer, do not
nothing
      while (data[prodslot] == data[conslot]);
      ops_wait(id);
      printf("Consuming item number %d...\n", data[conslot]);
      sleep(1);
      char item = data[data[conslot]];
      printf("Consumed item number %d.  Item value was %d\n",
             data[conslot], item);
      //Adds 1 to the position and mods prevents the consumer position from going beyond t
he buffer
      data[conslot] = (data[conslot] + 1) % bufferlength;
   }
    //Detatch
    shmdt(data);
    printf("All done consuming.\n");
    wait(&status); //For child process so that we can
    //Delete the shared memory
    printf("Child ended, removing shm\n");
    shmctl(shm_id, IPC_RMID, NULL);
 }
  else
  {
    //P2 - PRODUCER
    shm_id = shmget(ftok("prodcon_example2.c", 2), 0, 006);
    //Attach the shared buffer
    data = shmat(shm_id, (void *)0, 0);
    //Initialise the positions of the consumer & producer to 0
    data[conslot] = 0;
    data[prodslot] = 0;

    while (1){ //Runs the code indefinitely
      //Blocks the producer from moving on if the next position is the consumer.
      rsleep(); //Makes the producer sleep for a random amount of time
      while ((data[prodslot] + 1) % bufferlength == data[conslot]);
      printf("Producing item number %d...\n", data[prodslot]);
      sleep(2);
      data[data[prodslot]] = data[prodslot] * 2; //Simple data, easy to check.
      printf("Produced item number %d.  Value is %d\n",
             data[prodslot], data[data[prodslot]]);
      ops_signal(id);
      //Adds 1 to the position and mods prevents the producer position from going beyond t
he buffer
      data[prodslot] = (data[prodslot] + 1) % bufferlength;
   }
    //Detatch
    shmdt(data);
    printf("Producer finished.");
 }
```

```
}
```

BUFFER AT DIFFERENT LENGTHS

bufferlength = 6

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session16/lab16$ ./prodcon
Semaphore 17956997 initialized with path 'prodcon_example2.c'.
Producing item number 0...
Produced item number 0.  Value is 0
Producing item number 1...
Consuming item number 0...
Consumed item number 0.  Item value was 0
Produced item number 1.  Value is 2
Producing item number 2...
Consuming item number 1...
Consumed item number 1.  Item value was 2
Produced item number 2.  Value is 4
Producing item number 3...
Consuming item number 2...
Consumed item number 2.  Item value was 4
Produced item number 3.  Value is 6
Producing item number 4...
Consuming item number 3...
Consumed item number 3.  Item value was 6
Produced item number 4.  Value is 8
Producing item number 5...
Consuming item number 4...
Consumed item number 4.  Item value was 8
Produced item number 5.  Value is 10
Producing item number 0...
Consuming item number 5...
Consumed item number 5.  Item value was 10
```

bufferlength = 4

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session16/lab16$ ./prodcon
Semaphore 17956997 initialized with path 'prodcon_example2.c'.
Producing item number 0...
Produced item number 0.  Value is 0
Producing item number 1...
Consuming item number 0...
Consumed item number 0.  Item value was 0
Produced item number 1.  Value is 2
Producing item number 2...
Consuming item number 1...
Consumed item number 1.  Item value was 2
Produced item number 2.  Value is 4
Producing item number 3...
Consuming item number 2...
Consumed item number 2.  Item value was 4
Produced item number 3.  Value is 6
Producing item number 0...
Consuming item number 3...
Consumed item number 3.  Item value was 6
Produced item number 0.  Value is 0
Producing item number 1...
Consuming item number 0...
Consumed item number 0.  Item value was 0
Produced item number 1.  Value is 2
Producing item number 2...
Consuming item number 1...
^C
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session16/lab16$
```

//Because the consumer and producer slots are stored relative to the buffer length, ie bufferLength + 1 or +2, a change in bufferlength is easy and will not need to reallocate a specific slot for the producer & consumer.

Rsleep() on the consumer, consumer sleeps for a random amount of time

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session16/lab16$ ./prodcon
Semaphore 17956997 initialized with path 'prodcon_example2.c'.
Sleeping for 5 secs
Producing item number 0...
Produced item number 0.  Value is 0
Producing item number 1...
Produced item number 1.  Value is 2
Consuming item number 0...
Consumed item number 0.  Item value was 0
Sleeping for 3 secs
Producing item number 2...
Produced item number 2.  Value is 4
Consuming item number 1...
Consumed item number 1.  Item value was 2
Sleeping for 5 secs
Producing item number 0...
Produced item number 0.  Value is 0
Consuming item number 2...
Consumed item number 2.  Item value was 4
Sleeping for 5 secs
Producing item number 1...
Produced item number 1.  Value is 2
Consuming item number 0...
Consumed item number 0.  Item value was 0
```

//Because this script utilizes a ring buffer, even if the consumer runs at a different speed as the producer, the producer will continue to produce while the consumer will consume only when the producer has finish producing, there by having no collisions. We can also see that the producer will only produce when the consumer has finished consuming that item.

Rsleep() on the producer, producer sleeps for random amount of time.

```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session16/lab16$ ./prodcon
Semaphore 17956997 initialized with path 'prodcon_example2.c'.
Sleeping for 5 secs
Producing item number 0...
Produced item number 0.  Value is 0
Sleeping for 3 secs
Consuming item number 0...
Consumed item number 0.  Item value was 0
Producing item number 1...
Produced item number 1.  Value is 2
Sleeping for 5 secs
Consuming item number 1...
Consumed item number 1.  Item value was 2
Producing item number 2...
Produced item number 2.  Value is 4
Sleeping for 5 secs
Consuming item number 2...
Consumed item number 2.  Item value was 4
Producing item number 3...
Produced item number 3.  Value is 6
Sleeping for 5 secs
```

//Because of our ring buffer, even though the producer is sleeping, the consumer will not consume until the producer is done producing. This also shows that a collision does not occur even when the producer and consumer are running at different speeds.

LAB ACTIVITY 17 TCP SERVER

A. BRIEF DESCRIPTION OF THE TCP SERVER ACTIVITY

This activity focuses on the TCP Server- Client Model. In this activity, we are required to successfully transmit data between a tcp server and client and extending that, to create a server that checks our baxter robot instructions if it is syntactically correct and or semantically correct.

The instructions are encoded by the client and sent to the server to be checked. The data is parsed by the server and the server performs the checks. The result is then encoded again and transmitted back to the client where the client can view the results.

B. COMMENTED CODE SHOWING SERVER.PY & CLIENT.PY

[Commented code here]

‘’’For this section, instead of using the pre-provided tcp-server and client, I referenced code available online and wrote my own version of a tcp client-server in Python.’’’

SERVER.PY

```
'''
This tcp server utilises the socket module and encodes/decodes data with the pickle
module during transmission. The server functions as a mediator for our baxter robot.

Baxter robot that takes in min of 2 arguments, max of 4 arguments which are
from a predefined instruction set and checks if the instruction is in a correct
instruction syntax/semnatics. It then returns an output if the instruction given
was understood or failed.
'''
import socket
import sys
import pickle #used for data encoding/decoding during transmission

host = 'localhost'
port = 1234
data_payload = 4096
#Arbitary value that can be changed, I used a high byte value so no data is cut off.

# Instruction Set Library
time = ['1second','2seconds','5seconds','unlimited']
move = ['left','right','forward','backward','stop']
objects = ['orange','apple','car','bus','diamond']
action = ['recognise','eat','see','lift','drop','fetch']
size = ['small','big','little','massive']
location = ['door','kitchen','table']

# Grouping combos based on row, ie. first row only has obj, action, time
combo_0 = [objects, action, time]
combo_1 = [objects,size,action]
combo_2 = [move,time]
```

```
combo_3 = [move,time,move,time]
combo_4 = [location,action,objects]

#Necessary as lists with vars cannot be printed as literal strs
combo_lib = [["move","time"],["location","action","object"],["object","action","time"],
             ["objects","size","action"],["move","time","move","time"],]

#Grouping instructions together as main lib
full_library = time + move + objects + action + size + location

# Checks which combination the user is inputing and attaches the str of combo chosen.
def baxter(instruction_set):
    combo=[]
    combo_str = []
    if len(instruction_set) < 2 or len(instruction_set) > 4:
        return("Error #0 - 2 to 4 arguements required!")
    for inst in instruction_set:
        if inst not in full_library:
            return("Error #1 - Instruction is not in library")
    if len(instruction_set) == 2:
        combo = combo_2
        combo_str = combo_lib[0]
    elif instruction_set[0] in location:
        combo = combo_4
        combo_str = combo_lib[1]
    elif instruction_set[0] in objects:
        if instruction_set[1] in action:
            combo = combo_0
            combo_str = combo_lib[2]
        else:
            combo = combo_1
            combo_str = combo_lib[3]
    else:
        combo = combo_3
        combo_str = combo_lib[4]
    return (is_syntactically_right(instruction_set,combo,combo_str))

# Checks if the combo inputted matches the instruction lib
# to see if the instruction is syntactically correct
def is_syntactically_right(instruction_set,combo,combo_str):
    for i in range(len(instruction_set)):
        if instruction_set[i] not in combo[i]:
            return(f"Syntax is {combo_str}\nError #2 - Instruction syntactically incorrect
")
    return(is_semantically_right(instruction_set))

def is_semantically_right(instruction_set):
    '''Function to check if robot is performing semantically right instructions
    These may be instructions that are syntactically right but are dangerous for our robot
. '''
    large_objects = ['car','bus']
```

```
    inedible_objects = ['diamond','car','bus']
    dangerous_actions = ['lift','drop','fetch']
    large_size = ['big','massive']
    semantically_right = True
    if 'eat' in instruction_set and any(item in inedible_objects
      for item in instruction_set):
        semantically_right = False
    elif any(item in dangerous_actions for item in instruction_set) and
      any(item in large_objects for item in instruction_set):
            semantically_right = False
    elif 'kitchen' in instruction_set and any(item in dangerous_actions
      for item in instruction_set):
        semantically_right = False
    elif any(item in location for item in instruction_set) and
      any(item in large_size for item in instruction_set):
        semantically_right= False

    if semantically_right == True:
        return ("Instruction syntactically & semantically correct")
    else:
        return ("Instruction syntactically correct, but semantically incorrect")


'''End of baxter robot'''

def bytes_to_list(data):
    #Decodes the bytes from the tcp client into a list object
    #by using the pickle module
    instruction_set = pickle.loads(data)
    return baxter(instruction_set)

def server(port):
    '''Server takes in list as input, parses it with pickle and performs the checks with
      ourbaxter robot syntax checker. It the returns if the instruction is understood or
      failed and encodes it back to bytes with pickle to be transmitted back to
      the client.

    A connection is closed immediately after an instruction is checked.'''
    #Creates a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (host, port)
    # Allows the reuse of the same address and port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    print ("Starting up baxter server on %s port %s" % server_address)
    #Binds the socket to the port
    sock.bind(server_address)
    # Listens or any incoming connections from clients
    sock.listen(1)
    while True:
        print ("Waiting to receive message from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
```

```
        if data:
            print ("Data: %s" %data)
            output = bytes_to_list(data) #decodes data to a list obj
            client.send(pickle.dumps(output)) #encodes the data and sends it back
            print ("sent %s back to %s" % (output, address))
        #Ends connection after everything is done
        client.close()

if __name__ == '__main__':
    server(port)
```

CLIENT.PY

```
'''
TCP client that utilises the socket module to transmit data to the server.
The client takes in an instruction list as input and returns if the instruction given to
out baxter robot is understood or failed.
'''
import socket
import sys
import pickle
import time

host = 'localhost'
port = 1234
def baxter_client(message):
    #Creates a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #Connects the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)

    #Sends data to our baxter server
    try:
        # Send data after encoding to bytes with pickle
        print ("Sending %s" % message)
        sock.sendall(pickle.dumps(message))
        # Wait for the response with arbitary length, used 100 so no data is lost.
        data = sock.recv(100)
        print ("Received: %s" % pickle.loads(data))
    except socket.error as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()

def is_list(message):
    if type(message) != list:
```

```
        print("Your instruction has to be a list!")
        print("Correct syntax: ['instruction1','instruction2']...")
    else:
        return baxter_client(message)

def tests():
    #Test that iterates throught the examples to see if the desired outcome is procuded
    test_lib = [['orange','see','1second'],['table','lift','diamond'],['drop','drop'],
    ['left', '2seconds'],['apple','small','eat'],['left','2seconds','forward','1second'],
    ['kitchen','ball','2seconds'],['bus','lift','5seconds'],['car','eat','2seconds'],['lef
t','2seconds','forward','1second'],
    ['apple','small','eat'],['table','lift','diamond'],]

    for i in range(len(test_lib)):
        is_list(test_lib[i])
        print("---------------")
        time.sleep(1) #pauses between transmitting new instructions

if __name__ == '__main__':
   tests()
```

C. EXAMPLE OF TCP SERVER TO CHECK ROBOT INSTRUCTIONS.

[Screenshots showing server doing sentence parsing]

```
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x04\x00\x00\x00leftq\x01X\x08\x00\x00\x002secondsq\x02X\x07\x00\
x00\x00forwardq\x03X\x07\x00\x00\x001secondq\x04e.'
sent Instruction syntactically & semantically correct bytes back to ('127.0.0.1', 39984)
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x05\x00\x00\x00appleq\x01X\x05\x00\x00\x00smallq\x02X\x03\x00\x0
0\x00eatq\x03e.'
sent Instruction syntactically & semantically correct bytes back to ('127.0.0.1', 39986)
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x05\x00\x00\x00tableq\x01X\x04\x00\x00\x00liftq\x02X\x07\x00\x00
\x00diamondq\x03e.'
sent Instruction syntactically & semantically correct bytes back to ('127.0.0.1', 39988)
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x04\x00\x00\x00dropq\x01h\x01e.'
sent Syntax is ['move', 'time']
Error #2 - Instruction syntactically incorrect bytes back to ('127.0.0.1', 39994)
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x04\x00\x00\x00leftq\x01X\x08\x00\x00\x002secondsq\x02e.'
sent Instruction syntactically & semantically correct bytes back to ('127.0.0.1', 39996)
Waiting to receive message from client
Data: b'\x80\x03]q\x00(X\x05\x00\x00\x00appleq\x01X\x05\x00\x00\x00smallq\x02X\x03\x00\x0
0\x00eatq\x03e.'
sent Instruction syntactically & semantically correct bytes back to ('127.0.0.1', 39998)
```

[Screenshot of data received on client-side]

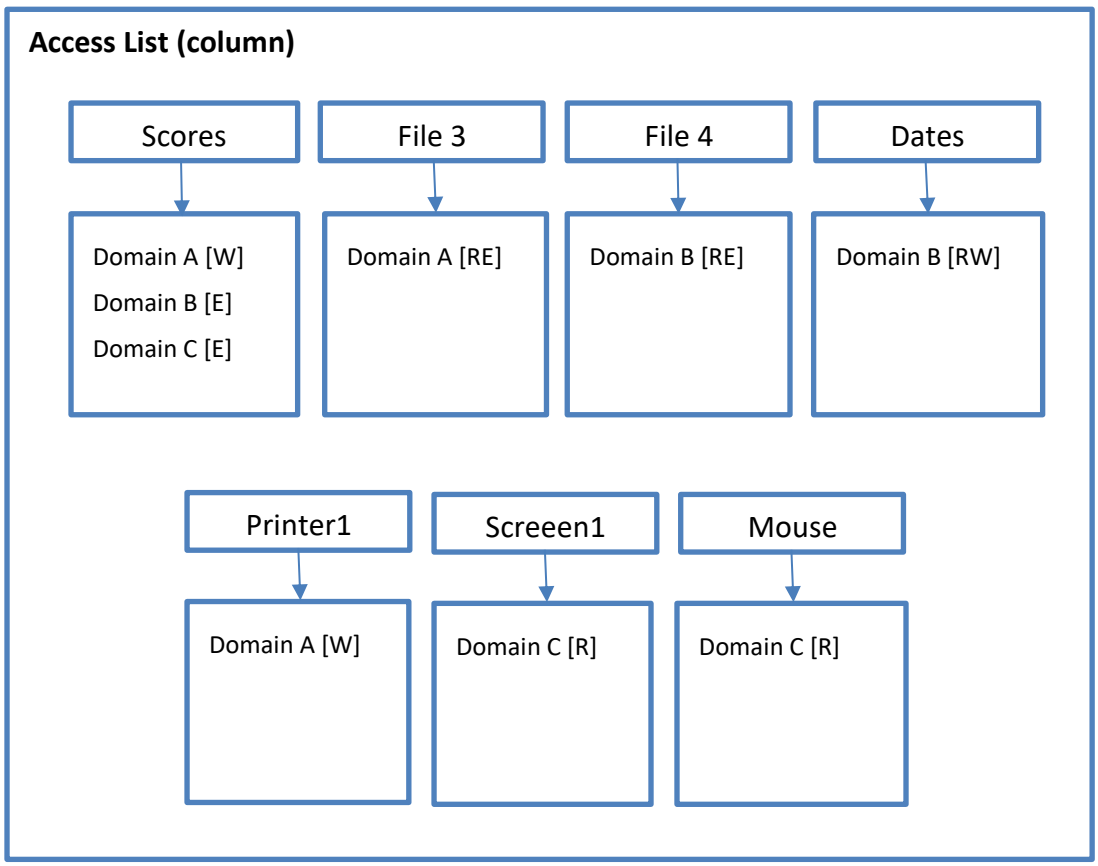
```
shawnhos@hvs-its-lnx01:~/207SE_Sessions/Session17/tcp-server$ python3 client.py
Connecting to localhost port 1234
Sending ['drop', 'drop']
Received: Syntax is ['move', 'time']
Error #2 - Instruction syntactically incorrect
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['left', '2seconds']
Received: Instruction syntactically & semantically correct
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['apple', 'small', 'eat']
Received: Instruction syntactically & semantically correct
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['left', '2seconds', 'forward', '1second']
Received: Instruction syntactically & semantically correct
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['kitchen', 'ball', '2seconds']
Received: Error #1 - Instruction is not in library
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['bus', 'lift', '5seconds']
Received: Instruction syntactically correct, but semantically incorrect
Closing connection to the server
--------------
Connecting to localhost port 1234
Sending ['car', 'eat', '2seconds']
Received: Instruction syntactically correct, but semantically incorrect
Closing connection to the server
```

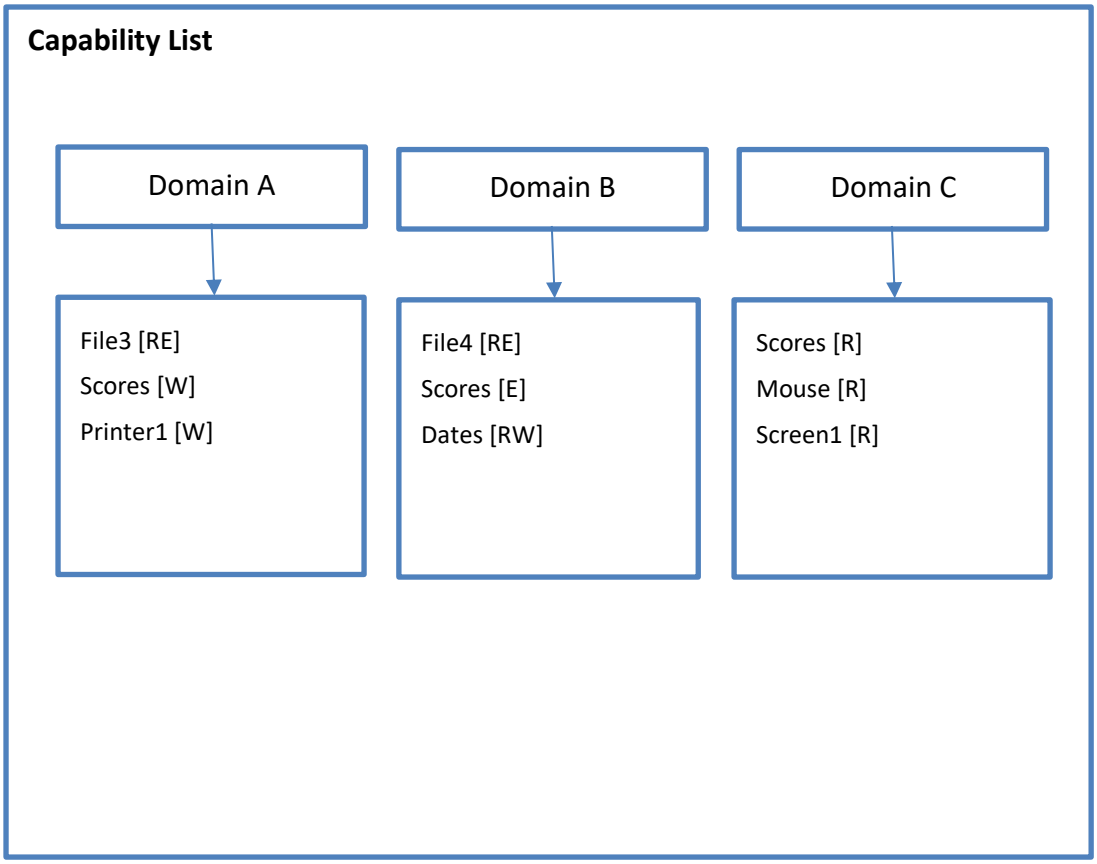
LAB ACTIVITY 19 SECURITY

a. CREATE A PROTECTION DOMAIN MATRIX, ACCESS LIST AND CAPABILITY LIST FOR THE DIAGRAM BELOW

[Diagrams here]

PROTECTION DOMAIN MATRIX							
	Scores	File3	File4	Dates	Printer1	Screen1	Mouse
DA	W	RE			W		
DB	E		RE	RW			
DC	E					R	R





B. COMMENTED CODE SHOWING UNIQUE HASH FUNCTION WITH SALT

[Commented code here]

```
'''
This salt + hash function takes any str password and returns an 18-char hexadecimal hash.
The salt is a generated by a the randint() function which creates a 128 digit number.
The hash utilises multiplication, addition, XOR and random prime numbers.
'''
import random
random_prime=10853082333529 #Can be any prime.

def salt(password):
    '''Creates a random 128 digit number that is appended to the password together with th
e length of the password'''
    salt=random.randint(10**127,(10**128)-1)
    #Uses the randint module to generate a 128 digit number.
    salted_password=str(password) + str(salt) #Concatenates the password and its salt
    salted_password = str(salted_password) + str(len(salted_password)) #We append the leng
th of the password to the end of the str
    return salted_password #The result is a long random str

def str_to_num(salted_password):
    '''Converts the input salted password into its ascii values & adds padding if necessar
y'''
    num_list = []
    while (len(salted_password) % 32 != 0):
```

```
        # Add padding to the password to make it a multiple of 32 (could be a multiple of
any number).
        salted_password = salted_password + str(0)
    for i in salted_password:
        i=ord(i) #Converts each char to its ascii value
        num_list.append(i) #Appends this value to our number list
    return num_list

def hash19(num_list):
    '''
    Takes the list of ascii values and returns a fixed length number represented in hex
    '''
    total = 0
    if len(num_list) % 2 != 0:
        # If the number of elements in our list is not even, we will append '1' to make it
 even
        num_list.append("1")
    for i in range(0,len(num_list),2):
        # Goes through the list in steps of 2, multiplying adjacent numbers and adding it
to the total
        sum = (num_list[i]*num_list[i+1])
        total+= sum
    # The result is multiplied with the rand prime and XORED with the total.
    output = (total*random_prime)^total
    return hex(output) #Returns the hashed value in hexadecimal

def crypt(password):
    #password = input("What do you want to hash?\n ")
    hashed = hash19(str_to_num(salt(password)))
    return(f"Your hash for {password} is {hashed} which is {len(str(hashed))} chars long!"
)

def test():
    '''Simple test of values of different lengths & similar inputs'''
    test_list = ["1 ","1","shawnhoo","shawnhoo","shawnhos","supercalifragilisticexpialidoc
ious"]
    for i in test_list:
        print(crypt(i))

test()
```

[outcomes of code here]

```
C:\Users\Hoo\PycharmProjects\hash\venv\Scripts\python.exe C:/Users/Hoo/PycharmProjects/hash/venv/Scripts/labi
Your hash for 1  is 0x1f64ee29a29e68f0 which is 18 chars long!
Your hash for 1 is 0x207ed3bbed8689b8 which is 18 chars long!
Your hash for shawnhoo is 0x25e6704543697c38 which is 18 chars long!
Your hash for shawnhoo is 0x259f7df5cbc334d8 which is 18 chars long!
Your hash for shawnhos is 0x2661c906cc5e6280 which is 18 chars long!
Your hash for supercalifragilisticexpialidocious is 0x3cfc175fd98dafa0 which is 18 chars long!

Process finished with exit code 0

Console  Terminal  4: Run  5: Debug  6: TODO
```

REFERENCES

1. Abualrob 2012 *Difference between Multiprogramming, Multitasking, Multithreading and Multiprocessing* [online] available from < https://www.8bitavenue.com/difference-between-multiprogramming-multitasking-multithreading-and-multiprocessing>

2. Lithmee (2018) *Difference Between Job Scheduling and CPU Scheduling* [online] available from <https://www.differencebetween.com/difference-between-job-scheduling-and-vs-cpu-scheduling/ >

3. Tolomei.G (n.d) *Multiprogramming, Multiprocessing, Multitasking, and Multithreading* [online] available from <https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/multiprogramming-multiprocessing-multitasking-multithreading/ >

4. Wikipedia (2020) *Multithreading (computer architecture)* [online] available from <https://en.wikipedia.org/wiki/Multithreading_(computer_architecture) >

5. Zhang Y., Zhang J. and Zhang D., (2009) "Implementing and Testing Producer-Consumer Problem Using Aspect-Oriented Programming," *Fifth International Conference on Information Assurance and Security, Xi'an,* [online] 749-752. Available from < https://ieeexplore.ieee.org/document/5284113>