



OPERATING SYSTEMS FINAL PROJECT REPORT



Rushi Patel
Yash Patel
Jeet Desai

1. How many processes can be in the running/executing state simultaneous (number of threads).

- This depends on the number of cores a laptop has. As the cores on the laptop increases the number of threads that can run simultaneously increases. For this project there are three threads that need to run. However, for the cases of thread 1 and thread 2, they need to run simultaneously and only after the execution of the first two threads, thread 3 can finally run.

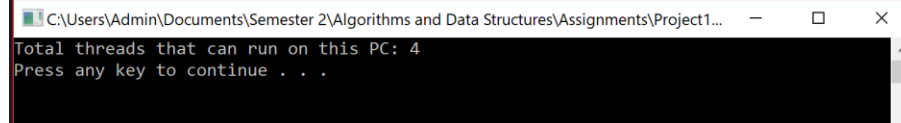
```
#include <iostream>
#include <thread>
using namespace std;

int main() {

    //may return 0 when not able to detect
    unsigned concurrentThreadsSupported = std::thread::hardware_concurrency();

    cout << "Total threads that can run on this PC: " << concurrentThreadsSupported << endl;

    system("PAUSE");
    return 0;
}
```



Here is a test that was done on one of our laptops in visual studio c++ to check how many threads can run concurrently.

2. How does the operating system use interrupts? (interrupts methods)

- Interrupts are essentially signals that are sent to the CPU by I/O devices. They notify the CPU to halt its current activities and execute the signal. In this project we are using a software interrupt where we are telling the OS to perform a system call requested by the program. The system call is communication which signals the program that priority packet 2 is detected.

```
//SignalHandler to make sure that processor receives a signal
void signalhandler(int priority)
{
    signalled = 1;
}

//Register a signal
void registersignal()
{
    //Installing a single handler
    signal(SIGINT, signalhandler);
    raise(SIGINT);

    //If signal is detected
    if (signalled)
    {
        cout << "Priority 2 packet has been detected " << endl;
    }
    else
    {
        cout << "Signal is not handled " << endl;
    }
}
```

The snippet above is our signal implementation. To implement this, we first had to define the signal class "#include signal.h". We are using SIGINT (Signal Interrupt) which is created by the user. In the registerSignal() function we are creating a signal if the condition of if/else statements match. If it's true it is sent to where thread 2 is functioning. If it's not true, a signal state that the signal was not handled.

```

//checking for vlan 100 packet that has a priority of 2
if ((packet.getVlan()) == '0' && (packet.getPriority()) == '2')
{
    pq100.insertHeap(FIFO.Front()); //if true, add packet from FIFO queue to this priority queue
    pq100.bubbleUp(); //heapify packet in the queue

    registersignal(); //signal to the screen that priority detected for 101
}

```

In the snippet above, we are calling the “registerSignal()” function in the if statement. The if statement checks if the packet in the queue is part of 100 and also if it has a priority of 2. If it has priority of 2, use the signal function to signal (print) to the screen/log file that priority packet is detected.

3. Why are system calls needed? Try to think of the benefits on all levels (dual-mode operation).

- The system calls are needed so users do not have to code programs to get basic functionality from the OS. The user can simply use the system call to request services from the operating system. There are total number of five system calls which are related to this project. Five system calls are process control, file management, device management, Information maintenance, and communication. These systems calls are used according to their requirements at specific part of this project.
- Process control:
 - Works in kernel level as CPU sends a request to kernel level to process all of the threads
 - These process would be the 3 threads which were created
 - Kernel level also used to signal from thread 1 to thread 2 and terminates thread 1 and thread 3 once they are finished executing

```

//main function
int main()
{
    //start time
    //clock_t starttime = clock();

    //initialize thread 1 and join part1 function to execute
    thread number1(part1);
    number1.join(); //executing thread 1

    //end time
    //clock_t endtime = clock();

    //calculate total elapsed time of thread 1
    //long double elapsedtime = (long double(endtime - starttime) / (CLOCKS_PER_SEC));

    //initialize thread 3 and join part3 function to execute
    thread number3(part3);
    number3.join(); //executing thread 3

    //cout << "Elapsed Time: " << elapsedtime << endl;

    system("pause");
    return 0;
} //end main

```

- File management:
 - This section refers to a Log file which was used to Log timestamp, payload, source and destination mac address

```
//thread 2 implementation
void part2(string Smac, string Dmac)
{
    //locking the printing capability to screen and log file for one thread at a time on
    mtx.lock();

    // I/O object
    ofstream myfile;

    cout << "Packets with priority 2" << endl;

    //open log file to write to file
    myfile.open("log.txt", ios::in | ios::out | ios::ate);

    //set timestamp to allocate to each packet
    auto clock = chrono::system_clock::now();
    time_t time = chrono::system_clock::to_time_t(clock);
```

- Device management:
 - Used on user-level to print all the information regarding packets
 - This part will print all the information on the screen and log file

```
//write packet content to log file (timestamp, source mac address and destination mac address)
myfile << "Time: " << time << " | Source MAC Address: " << Smac << " | Destination MAC Address: " << Dmac << endl;

//write packet content to screen (timestamp, source mac address and destination mac address)
cout << "Time: " << time << " | Source MAC Address: " << Smac << " | Destination MAC Address: " << Dmac << " \n" << endl;
```

- Information maintenance:
 - Runs in kernel level since code is running inside a thread which is basically running in a kernel mode

- A packet class was created to store information regarding a packet

```
class packet
{
public:
    packet(); // Default Constructor

    //Defining all set functions
    void setSourceMAC(string);
    void setDestMAC(string);
    void setVlan(string);
    void setPriority(string);
    void setData(string);
    void setPayload(string);
    void setVlanNum(string);

    //Defining all get functions
    string getSourceMAC();
    string getDestMAC();
    char getVlan();
    char getPriority();
    string getData();
    string getPayload();
    string getVlanNum();

private:
    // Packet Variables
    string packetData;
    string payload;
    string vlanNum;
    char vlan;
    char priority;
    string sourceMAC;
    string destMAC;
    char array[80];
};
```

- Communication:
 - Refers to an interrupt where it is used to send a notification when a packet with priority 2 was detected

```
//SignalHandler to make sure that processor recieves a signal
void signalhandler(int priority)
{
    signalled = 1;
}

//Register a signal
void registersignal()
{
    //Installing a single handler
    signal(SIGINT, signalhandler);
    raise(SIGINT);

    //If single is detected
    if (signalled)
    {
        cout << "Priority 2 packet has been detected " << endl;
    }
    else
    {
        cout << "Signal is not handled " << endl;
    }
}
```

The snippet above is used to send signal to screen/log file when priority packet is detected.

References

“Programmatically Find the Number of Cores on a Machine.” c - Programmatically Find the Number of Cores on a Machine - Stack Overflow, stackoverflow.com/questions/150355/programmatically-find-the-number-of-cores-on-a-machine.

“1.12. System Calls.” *1.12. System Calls - Operating Systems Study Guide*, faculty.salina.k-state.edu/tim/ossg/Introduction/sys_calls.html.