SIT315 – Seminar 2 Real Time Systems

Group Members:

Sean Emery

Deol Satish

Leo Luong

Anno Gomes

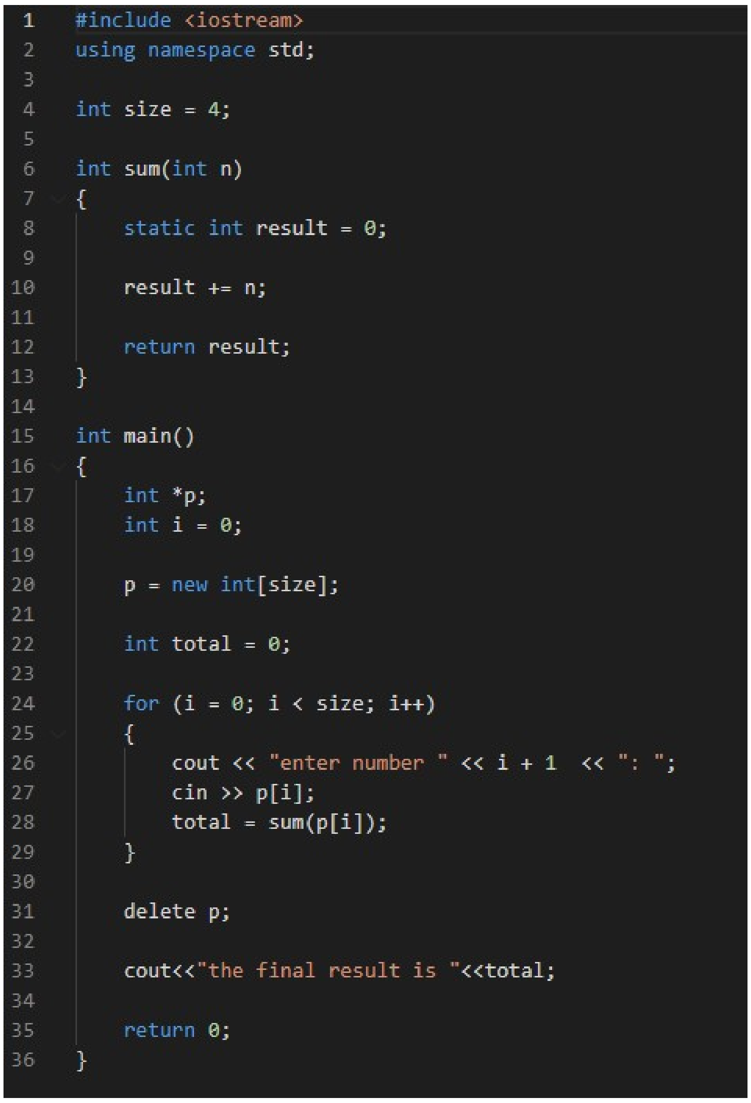
Asim Arshad

Gagan (Zoom name)

Jay (Zoom name)

Activity 1 – Process Memory Segments

Considering the C++ program below, and [4, 5, 8, 3] as user inputs,



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

int size = 4;

int sum(int n)
{
    static int result = 0;

    result += n;

    return result;
}

int main()
{
    int *p;
    int i = 0;

    p = new int[size];

    int total = 0;

    for (i = 0; i < size; i++)
    {
        cout << "enter number " << i + 1  << ": ";
        cin >> p[i];
        total = sum(p[i]);
    }

    delete p;

    cout<<"the final result is "<<total;

    return 0;
}
```

Answer the following questions:

1. **Identify what is stored in the text segment when the process is admitted by the OS and is in the ready state**.
   The compiled machine code of the program is stored in the text segment.

2. **Identify what variables will be stored in the data segment.**
   Global and static variables will be stored in the data segment, in this case size and result.

3. **Identify what is stored in heap and stack segments when line 12 is being executed for the third time.**
   The heap will have the array for p, which when line 12 is executed for the third time will be [4, 5, 8, 0/NULL]. The stack will have the value of n as 8 and result as 17 for the sum function, and i as 2, total as 9 (before 17 is returned) for the main function and a pointer to p.

4. **Identify what is stored in heap and stack segments when line 33 is being executed.**
   The heap will still have the array for p, which will now be [4, 5, 8, 3], but will not be pointed to. The garbage collector will clear this memory. The stack will have just the main function will i being 4, and total as 20.

Activity 2 – Interrupts

Based on the Tinkercad circuit accessible from the link below, complete the following activities:

https://www.tinkercad.com/things/3XukUuR7UtR

1. **Based on the design and the code, explain what the primary function of this board is. Complete the code by adding appropriate comments in the designated lines.**
   The board's primary function is to turn on an LED while a button is pushed. Code with full comments:

```
// Assign the button and LED pins as constants
const uint8_t BTN_PIN = 2;
const uint8_t LED_PIN = 13;

// set button and LED initial states to low
uint8_t buttonPrevState = LOW;
uint8_t ledState = LOW;

// Setup function ran once at start up
void setup()
{
 // Set the button pin as an input
 pinMode(BTN_PIN, INPUT_PULLUP);
 // Set the LED pin as an output
 pinMode(LED_PIN, OUTPUT);
 // Start the serial connection at 9600 baud rate
 Serial.begin(9600);
}

// Ran after the setup and loops forever
void loop()
{
 // Read the value of the button pin and store in the
 // button state
 uint8_t buttonState = digitalRead(BTN_PIN);

 // Print to the serial the button state, previous state
 // and LED state
 Serial.print(buttonState);
 Serial.print(buttonPrevState);
 Serial.print(ledState);
 Serial.println("");


 // If the button state has changed, change the LED state
 // and output the changed state to the LED pin
 if(buttonState != buttonPrevState)
 {
  ledState = !ledState;
  digitalWrite(LED_PIN, ledState);
 }

 // Set the current button state to previous button state
 // for next loop
 buttonPrevState = buttonState;

 // Pause the loop for 500 milli seconds
 delay(500);
}
```

2. **Identify what the main problem in the code is and how it can affect the end-users.**
   The main problem with the code is that if the button is pressed during the delay, the LED will not turn on. This could leave the user waiting up to half a second for the LED to respond and this wait would increase if the delay was increased.

3. **Change the code to resolve the problem you identified in 2.**
   I have added an interrupt to the code as seen here.

```
// Assign the button and LED pins as constants
const uint8_t BTN_PIN = 2;
const uint8_t LED_PIN = 13;

// set button and LED initial states to low
uint8_t buttonState = LOW;
uint8_t ledState = LOW;
```

```
// Setup function ran once at start up
void setup()
{
  // Set the button pin as an input
  pinMode(BTN_PIN, INPUT_PULLUP);
  // Set the LED pin as an output
  pinMode(LED_PIN, OUTPUT);
  // Start the serial connection at 9600 baud rate
  Serial.begin(9600);
  // Set up an interupt on the button pin when it changes
  attachInterrupt(digitalPinToInterrupt(BTN_PIN), button_ISR, CHANGE);
}

// Ran after the setup and loops forever
void loop()
{
  // Read the value of the button pin and store in the
  // button state
  buttonState = digitalRead(BTN_PIN);

  // Print to the serial the button state and LED state
  Serial.print(buttonState);
  Serial.print(ledState);
  Serial.println("");

  // Pause the loop for 500 milli seconds
  delay(500);
}

// Button Interupt Service Routine
void button_ISR()
{
  // change the LED state as the button state has changed
  ledState = !ledState;
  // Output the LEDs new state to the LED pin
  digitalWrite(LED_PIN, ledState);
}
```

Activity 3 – Process Scheduling

In the week one lecture, we have briefly discussed three scheduling algorithms, namely FCFS, Round-Robin, and Preemptive Priority-based scheduling

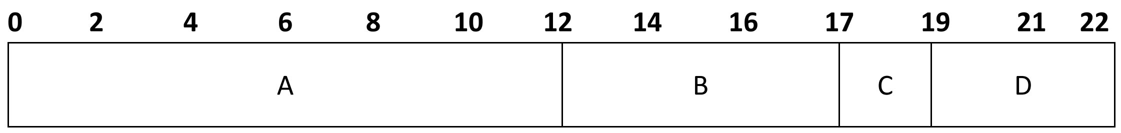
Based on the table below, answer the following questions:

Process	Arrival Time	Burst Time	Priority
A	0	12	2
B	1	5	4
C	3	2	1
D	4	3	3

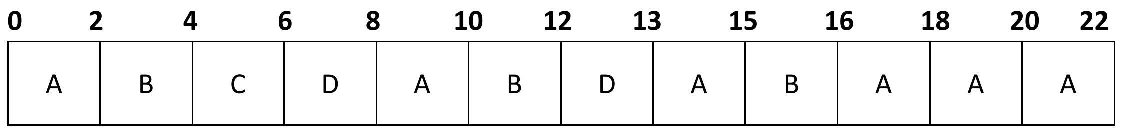
The quantum time is 2, which means each process is only executing for 2 units of time at a time.

1. **Draw three Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, Round-Robin, and Preemptive Priority-based scheduling.**

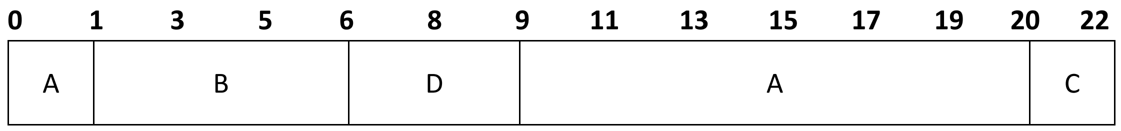
FCFS



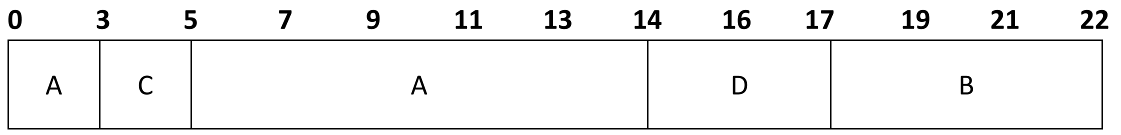
Round-Robin



Preemptive Priority-based (taking 4 as the highest priority)



Preemptive Priority-based (taking 1 as the highest priority)



2. **For each of the scheduling algorithms, compute the waiting times of each process.**
   Using the formula of process finish time – process arrival time – process burst time.

	FCFS	**Round-Robin**	**Preemptive Priority-based (4 high)**	**Preemptive Priority-based (1 high)**
Process	**Wait Time**	**Wait Time**	**Wait Time**	**Wait Time**
A	0	10	8	2
B	11	10	0	16
C	14	1	17	0
D	15	6	2	10

3. **Compute the average waiting time of each scheduling algorithm.**

FCFS	**Round-Robin**	**Preemptive Priority-based (4 high)**	**Preemptive Priority-based (1 high)**
Average Wait Time	**Average Wait Time**	**Average Wait Time**	**Average Wait Time**
0 + 11 + 14 + 15 = 41	10 + 10 + 1 + 6 = 28	8 + 0 + 17 + 2 = 27	2 + 16 + 0 +10 = 28
41 / 4 = 10	28 / 4 = 6.75	27 / 4 = 6.75	28 / 4 = 7

4. **Research and find another scheduling algorithm that has a lower waiting time than these three algorithms.**
   Shortest Job First (SJF) is considered as the lowest average wait time. This has the minimum waiting time amongst all the scheduling algorithms but has difficulty in implementation due to predicting the time of the next job.