

Asphalt

CSEN 602-Operating Systems Report

Team 35:

Khaled Moataz	40-5296
Omar El-Fatairy	40-9247
Mohamed Alaa	40-7699
Mohamed Atef	40-4541

Tutorial: T-14

First Draft Report

Table of Contents:

<i>Introduction</i>	3
<i>OS Type and Hardware</i>	3
<i>Features</i>	3
<i>1.Parking Sensors</i>	3
<i>2.Seatbelt safety warning</i>	4
<i>3.Cruise Control</i>	4
<i>4.Safety Speed Warning</i>	5
<i>Methodology</i>	5
<i>Basic features and functionality</i>	6

Introduction:

Asphalt is an operating system that aims for the safety and the comfort of the driver. Our main goal is to give drivers a multitude of easily accessible features, while providing real time information and preserving the safety of all those in the vehicle.

Operating System Type and Hardware:

The Operating System type will be Real Time Operating System since we need the functions of the car to be right on time without any possible delays for the safety of the driver and to assure the accessibility and comfort of the driver and the hardware used will be that of a car.

Features:

1. **Parking Sensors:** This feature is to ensure the safety of the driver and the surroundings while parking in order not to cause damage to his precious car or hit other people.

- **User Mode:** when the user goes in reverse mode, the parking sensors will be activated.
- **Kernel Mode:** there will be warning sensor that will emit a sound according to the safety distance at the back of the car and depending on the frequency of the sound the closer the object is.

2. **Seatbelt safety warning:** This feature is to make sure of the safety of the driver/passenger and to make sure that the driver abides by the law.

- **User Mode:** N/A
- **Kernel Mode:** When the OS sees that the driver is driving the car with speed of more than 20 km/hr for more than 10 seconds the car will initialize a warning siren and a flashing light to notify the driver that he is not buckled in and if the driver/passenger didn't wear the seatbelt after one minute the siren will get louder.

3. **Cruise Control:** This application will help the driver to ensure comfort of the driver while travelling for long distance since this application will give him/her the option to set the speed to a constant value and not using the gas pedal until further input.

- **User Mode:** The user will have the ability to set the speed he want to stay at as an input to the OS then the OS takes over.
- **Kernel Mode:** Takes over the throttle of the car to maintain a steady speed as set by the driver without the user using the gas/brake pedal.

4. Safety Speed Warning: This will ensure that the driver will always be aware if he crossed the speed of 120 km/hr.

- **User Mode:** N/A.
- **Kernel Mode:** Whenever the driver goes over the speed limit an alert sound will be emitted with a visual warning notifying the driver that he exceeding the speed limit.

Methodology:

We will be using Java programming language to simulate the operating system. The reason for choosing java is that we are already familiar with it, the operating system will be a single threaded system and will implement multiprocessing, the operating system will include I/O tables to control the I/O devices and an interrupt handler to handle interrupts, the interrupts will be prioritized based on the importance of the application to ensure the driver's safety and a better experience, The operating system will be simulated using the GUI in java.

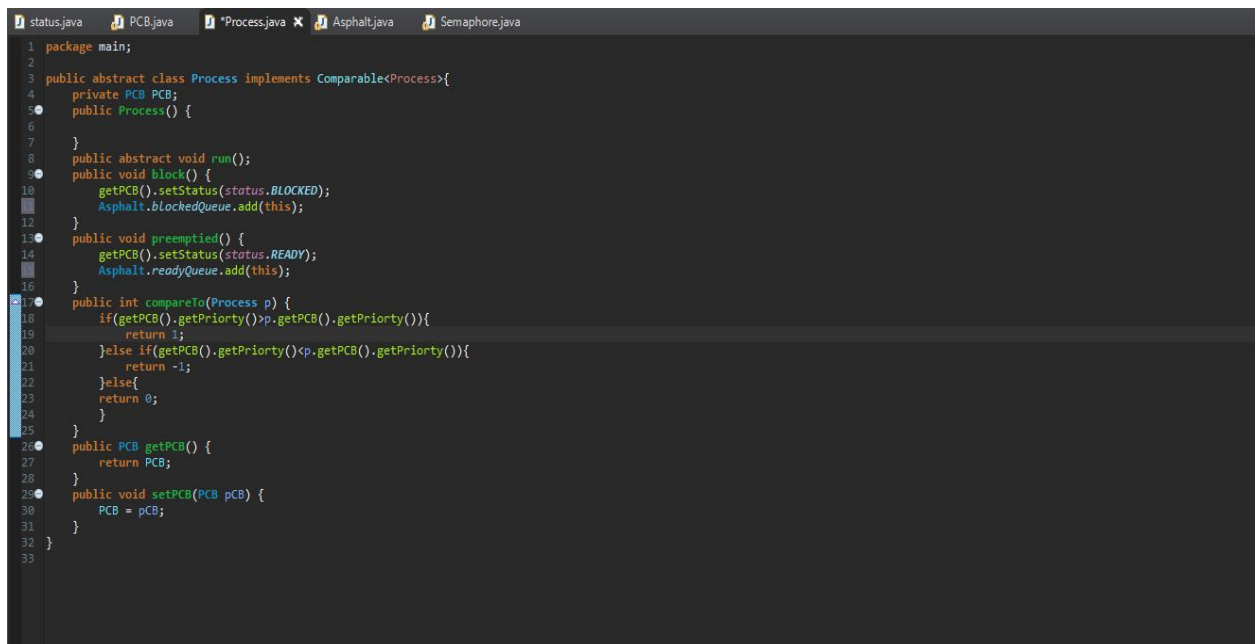
Basic Features and Functionalities:

A. Processes:

Each application has its own process and each process has its own PCB and the PCB contains all the information needed for the process to function and to be managed by the OS, and this will be implemented as follows:

The process class will be implemented as an abstract class and each application will inherit from the process class (each application is only one process) and each process will have a PCB and the PCB will have the process id, status, stack pointer and other needed information, the process class will also contain an abstract run method which will contain the code of each application, and also 2 methods (block and preempted) and here is a snippet of the code:

-The process code



```
1 package main;
2
3 public abstract class Process implements Comparable<Process>{
4     private PCB PCB;
5     public Process() {
6
7     }
8     public abstract void run();
9     public void block() {
10         getPCB().setStatus(status.BLOCKED);
11         Asphalt.blockedQueue.add(this);
12     }
13     public void preempted() {
14         getPCB().setStatus(status.READY);
15         Asphalt.readyQueue.add(this);
16     }
17     public int compareTo(Process p) {
18         if(getPCB().getPriority()>p.getPCB().getPriority()){
19             return 1;
20         }else if(getPCB().getPriority()<p.getPCB().getPriority()){
21             return -1;
22         }else{
23             return 0;
24         }
25     }
26     public PCB getPCB() {
27         return PCB;
28     }
29     public void setPCB(PCB pCB) {
30         PCB = pCB;
31     }
32 }
33
```

-The Process Control Block code

```
1 package main;
2
3 import java.util.ArrayList;
4
5 public class PCB {
6     private int id;
7     private int userID;
8     private status status;
9     private int stackPointer;
10    private int priority;
11    private int eventID;
12    private ArrayList<Integer> privileges;
13    private static int currentID=0;
14
15    public PCB(int userID,int priority,ArrayList<Integer> privileges) {
16        this.id=currentID++;
17        this.setStatus(getStatus().NEW);
18        this.userID=userID;
19        this.stackPointer=0;
20        this.priority=priority;
21        this.eventID=0;
22        this.privileges=privileges;
23    }
24
25    public int getPriority() {
26        return priority;
27    }
28    public int getID() {
29        return id;
30    }
31
32    public status getStatus() {
33        return status;
34    }
35
36    public void setStatus(status status) {
37        this.status = status;
38    }
39
40 }
41
```

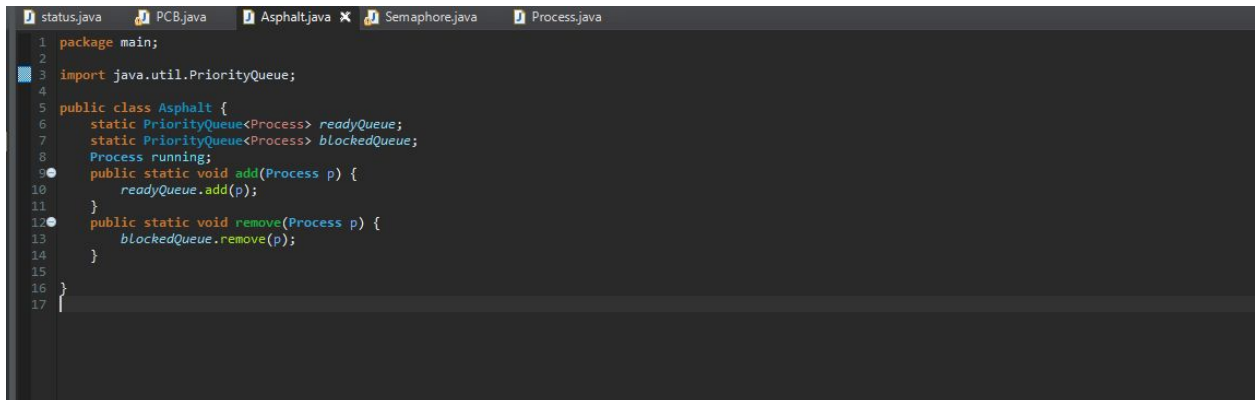
B.Semaphore:

We will be using Mutex to regulate the access of the critical section which is in the safety seatbelt, parking sensor, and safety speed warning which will access the critical resource (such as the alarm system) and the display which displays an icon that shows the warning.

```
1 package main;
2
3 import java.util.LinkedList;
4
5 public class Semaphore {
6     private boolean value;
7     private int ownerID;
8     private Queue waiting;
9
10    public Semaphore(boolean value) {
11        value = true;
12        waiting = new LinkedList<String>();
13    }
14
15    void semWait(Process p) {
16        if (value) {
17            ownerID = p.getPCB().getID();
18            value = false;
19        } else {
20            waiting.add(p);
21            Asphalt.add(p);
22        }
23    }
24
25    void semSignal(Process p) {
26        if (ownerID == p.getPCB().getID()) {
27            if (waiting.isEmpty()) {
28                value = true;
29            } else {
30                Asphalt.add((Process)waiting.remove());
31                Asphalt.remove(p);
32            }
33        }
34    }
35
36 }
37
38 }
39
40
```

C.Priorities:

We will be implementing a priority scheduling system that will prioritize the applications according to their level of importance. The OS will contain a priority Queue that handles this functionality using the `compareTo()` method that is implemented in the process class, here is the code:

A screenshot of a Java IDE with a dark theme. The top of the window shows several open files: status.java, PCB.java, Asphalt.java (which is the active file), Semaphore.java, and Process.java. The code in Asphalt.java is as follows:

```
1 package main;
2
3 import java.util.PriorityQueue;
4
5 public class Asphalt {
6     static PriorityQueue<Process> readyQueue;
7     static PriorityQueue<Process> blockedQueue;
8     Process running;
9     public static void add(Process p) {
10         readyQueue.add(p);
11     }
12     public static void remove(Process p) {
13         blockedQueue.remove(p);
14     }
15 }
16
17 }
```

D.Concurrency:

This will be implemented in the main class as intervals of clock cycles that will alternate between processes with the help of the stack pointer in the processes (PCB) that will point to the line which is to be executed.