Exercise 4

STUDENT NAME: ___Haruto Onoda_____ STUDENT ID: __s1270195_____

Ex.1

Answer the following questions and explain your answers.

a) What is the meaning of the term *busy waiting*?

A process that executes something or waits for execution time when a certain condition is met is called The process will be executed when it satisfies the

b) Is there a way to avoid busy waiting? If yes, explain the way. (Simple yes/no answer is invalid.)

In the case of semaphores, the solution is to associate a queue with each semaphore. In a single-processor computer On a single-processor computer, busy waiting can be solved by disabling interrupts during wait() and signal(). Busy waiting can be solved by disabling interrupts during wait() and signal(). In a multiprocessor environment, interrupts must be disabled on all processors. You must do this.

Ex.2

The Cigarette-Smokers Problems.

Consider a system with three smoker processes and one agent process that play the following roles:

The smoker:

- Each smoker continuously makes a cigarette and then smokes it.
- To make and smoke a cigarette, the smoker needs three ingredients: tobacco, paper, and matches.
- One of the smoker processes has unlimited paper, another has tobacco, and the third has matches.
- The smoker smokes a cigarette during indefinite time.

The agent:

- The agent has an infinite supply of all three materials.
- The agent randomly places two of different ingredients on the table.

Scenario:

After placing two ingredients by the agent, the smoker, who has the remaining ingredient, makes the following actions:

1. Picks up two ingredients,
2. Signals the agent.
3. Makes a cigarette and smokes.
4. After smoking finished, the smoker returns to wait ingredients again.

After obtaining a signal from smoker, the agent puts out another two of three ingredients, and the cycle repeats.

Task:

Write a pseudo program to synchronize the agent and the smokers' processes using the following primitives:

- **`wait(S);`** and **`signal(S);`** – **wait** and **signal** for a semaphore S (See lecture slides).
- `randNum = rand(i,j);` - Pick a random number from i to j.
- **`Put_match&paper(); Put_tobacco&match (); Put_tobacco&paper();`** – put two of three ingredients on table.
- **`Pick_up_match&paper(); Pick_up_tobacco&match(); Pick_up_tobacco&paper();`** – pick up two ingredients.
- **`Make&Smoke();`** - make and smoke a cigarette.

Semaphores:

Use the following semaphores in your program (every semaphore is initialized to 1):

- `table` – to lock/unlock a table on which ingredients are put by the agent
- `agent` – to see that ingredients were picked up by one of the smokers
- `smoker_tobacco` – semaphore for the smoker who has a tobacco
- `smoker_paper` – semaphore for the smoker who has a paper
- `smoker_match` – semaphore for the smoker who has a match

```
----- Agent
repeat {
  wait (table);
  randNum = rand( 1, 3 ); // Pick a random number from 1 to 3
```

```
  if (randNum == 1) {
    Put_match&paper();
   signal(smoker_tobacco);
  } else if (randNum == 2) {
  Put_match&paper();
 signal(smoker_tobacco);
  } else if (randNum == 3) {
    Put_tobacco&paper();
    signal(smoke_match);
 }
  signal(Tabel)
  wait(agent); // Agent sleeps
} until false // end forever loop

----- Smoker 1 who has tobacco and needs match&paper

repeat {
 wait(smoker_tobacco);
 wait(table);
  wait(agent);
 Make&Smoke();
} until false // end forever loop

----- Smoker 2 who has paper and needs tobacco&match

repeat {
 wait(smoker_paper);
  Pick_up_tobacco&match();
  signal(Table);
  signal(agent);
 Make&Smoke();
} until false // end forever loop

----- Smoker 3 who has match and needs tobacco&paper

repeat {
 wait(smoker_match);
  Pickup_tobacco&paper();
  signal(Tabel);
  signal(agent);
 Make&Smoke();
} until false // end forever loop
```

Ex.3

Show that your program for Ex.2 satisfies the Critical-Section requirements (mutual exclusion, progress, and bounded waiting).

- Mutual exclusion

An if-statement control that executes only a process and not other processes at the same time.

- Progress

When an agent process finishes a critical section, the other processes wait in the wait() function.

- Boundary wait

Limits the number of times a process can be selected before entering the critical section, and signals that the process has exited with the signal() function.