**OPERATING SYSTEMS**

**UCCD 2103**

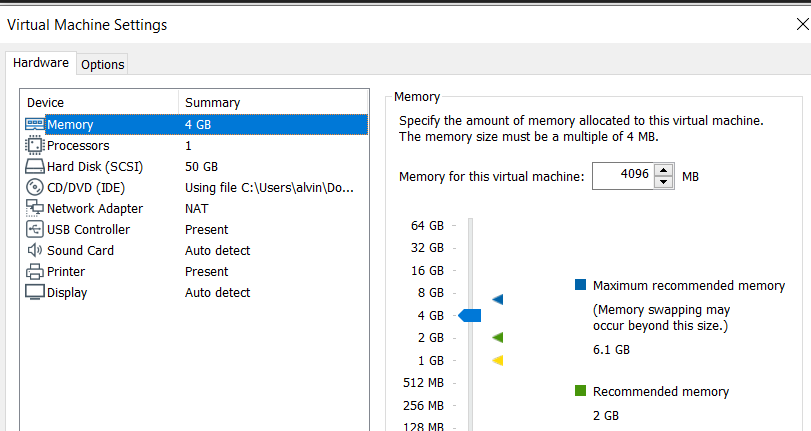
**JAN 2020**

**ASSIGNMENT REPORT**

**PART A**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Student ID | Name | Programme (CN/CS/CT/IA) | Email |
| 1. | 1806352 | Ewe Chun Kit(L) | CS | jackzewe120@1utar.my |
| 2. | 1806718 | Tai Jia Wei | CS | - |
| 3. | 1806623 | Wan Jia Xing | CS | - |

**VMWare Specification**

****

Processor: Uni Core

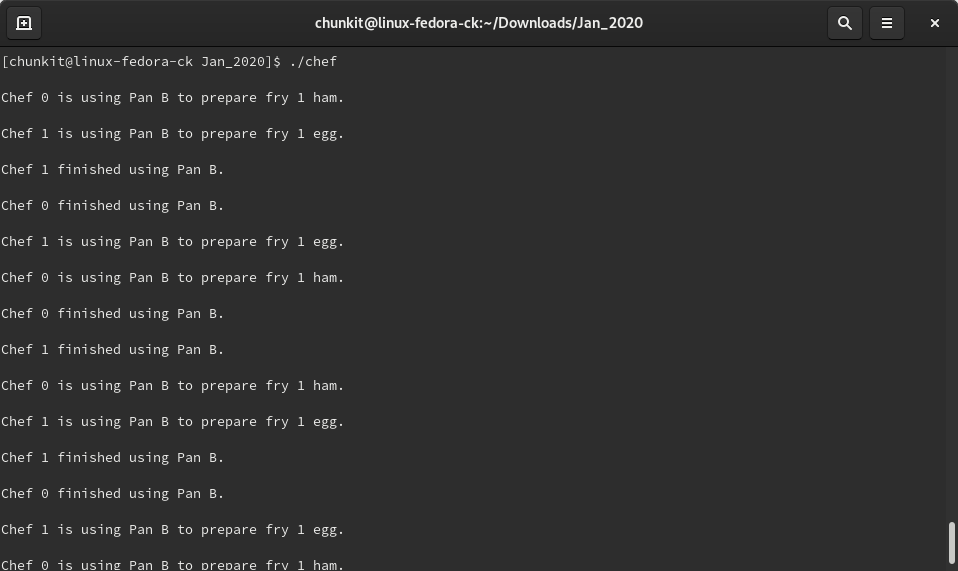
OS: Fedora

RAM allocated: 4GB

**Question 1**

|  |  |  |
| --- | --- | --- |
| t | thread\_t0() | thread\_t1() |
| 0 |  |  |
| 1 |  | printf(“\nChef 1 is using Pan B to prepare fry 1 egg.\n");  sleep(2); |
| 2 | printf(“\nChef 0 is using Pan B to prepare fry 1 ham.\n");  sleep(2); |  |
| 3 |  | printf("\nChef 1 finished using Pan B.\n");  sleep(1); |
| 4 | printf("\nChef 0 finished using Pan B.\n");  sleep(1); |  |
| 5 |  | printf(“\nChef 1 is using Pan B to prepare fry 1 egg.\n");  sleep(2); |
| 6 | printf(“\nChef 0 is using Pan B to prepare fry 1 ham.\n");  sleep(2); |  |

Timing Diagram



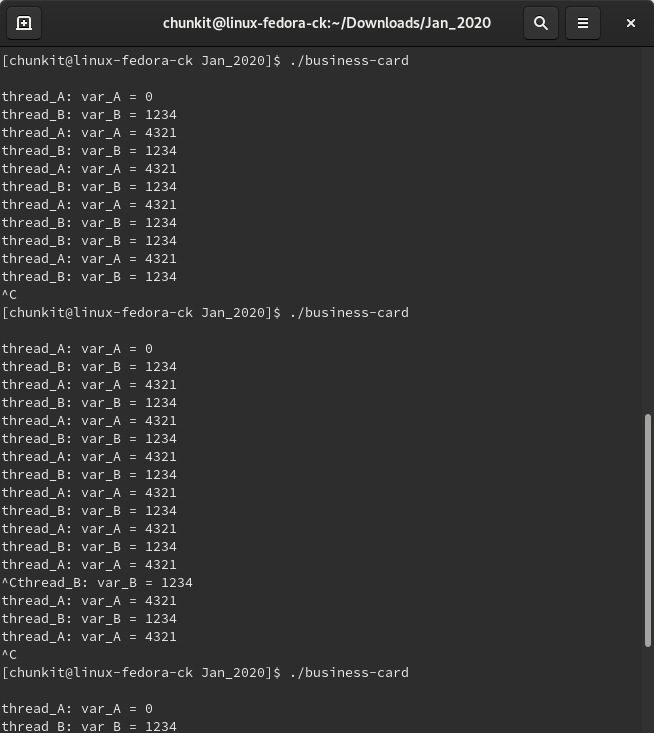
Output Diagram

As we can observe from the timing diagram above, Chef 1 and Chef 2 are doing the same activity at the same time. When the system execute thread 1 first output, the time slice coincidentally finished before it can finish its execution and it proceed to run thread 0. This is not the ideal output that we need and the reason is due to the lack of implementing semaphore in this system. Both thread is having collision of critical section and thus executing the same line. The output can be seen from the picture above.

**Question 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| t | thread A( ) | thread B( ) | A.count | B.count | var\_A | var\_B |
| 0 |  |  | 0 | 0 | 0 | 0 |
| 1 |  | var\_B=4321; sem\_post(&A); sem\_wait(&B); | 1 | -1 | 0 | 4321 |
| 2 | var\_A=1234; sem\_post(&B); sem\_wait(&A); buf\_A = var\_A; var\_A = buf\_B; printf(var\_A); sleep(1); |  | 0 | 0 | 1234 > 0 | 4321 |
| 3 |  | buf\_B = var\_B; var\_B = buf\_A; printf (var\_B);  sleep(1); | 0 | 0 | 0 | 1234 |
| 4 | var\_A=1234; sem\_post(&B); sem\_wait(&A); |  | -1 | 1 | 1234 | 1234 |
| 5 |  | var\_B=4321; sem\_post(&A); sem\_wait(&B); buf\_B = var\_B; var\_B = buf\_A; printf(var\_B);  sleep(1); | 0 | 0 | 1234 | 4321 > 1234 |
| 6 | buf\_A = var\_A; var\_A = buf\_B; printf(Var\_A); sleep(1); |  | 0 | 0 | 4321 | 1234 |

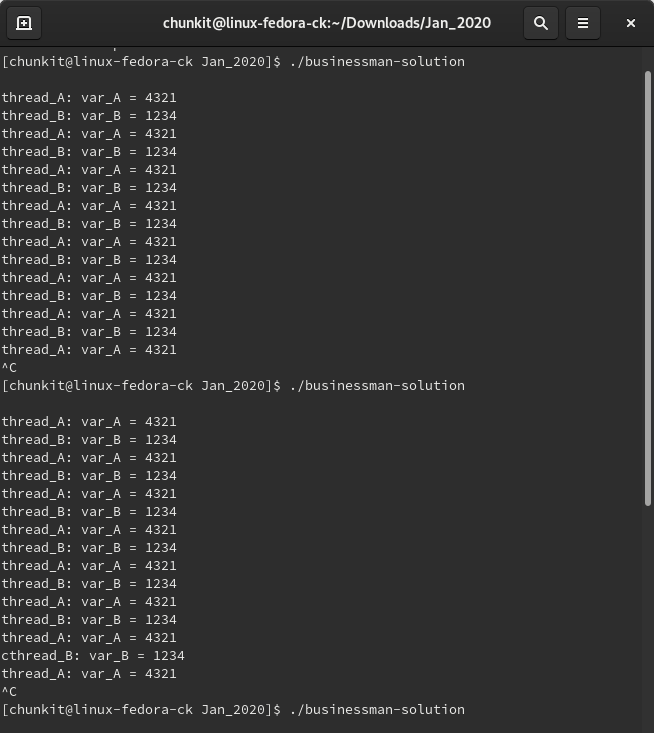
Timing Diagram



Output Diagram

Regardless of which thread is running first. During the first iteration, the output of the second thread that execute will always be zero. Referring to the timing diagram above, thread A() will be the one facing this problem. This is due to the arrangement of code (buf\_B = var\_B) in t=3 being placed wrongly and thus the buf\_B couldn’t able to run because semaphore wait has already run. Therefore, it went to block queue and in t=2 var\_A couldn’t take in any value because buf\_B haven’t initiate yet.

**Question 2 (Solution)**

****

Output Diagram

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| t | thread A( ) | thread B( ) | A.count | B.count | var\_A | var\_B |
| 0 |  |  | 0 | 0 | 0 | 0 |
| 1 |  | var\_B=4321; buf\_B = var\_B; sem\_post(&A); sem\_wait(&B); | 1 | -1 | 0 | 4321 |
| 2 | var\_A=1234; sem\_post(&B); sem\_wait(&A); buf\_A = var\_A; var\_A = buf\_B; printf(Var\_A); sleep(1); |  | 0 | 0 | 1234 > 4321 | 4321 |
| 3 |  | var\_B = buf\_A; printf(var\_B);  sleep(1); | 0 | 0 | 4321 | 1234 |
| 4 | var\_A=1234; buf\_A = var\_A; sem\_post(&B); sem\_wait(&A); |  | -1 | 1 | 1234 | 1234 |
| 5 |  | var\_B=4321; sem\_post(&A); sem\_wait(&B); buf\_B = var\_B; var\_B = buf\_A; printf (var\_B);  sleep(1); | 0 | 0 | 1234 | 4321 > 1234 |
| 6 | var\_A = buf\_B; printf(var\_A); sleep(1); |  | 0 | 0 | 4321 | 1234 |

Timing Diagram

**Appendix A (chef-solution.c)**

// Reference: https://www.geeksforgeeks.org/use-posix-semaphores-c/

// C program to demonstrate a simple multi-threaded process using two threads.

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <semaphore.h>

sem\_t s0;

//thread t0 represents Chef 0

void\* thread\_t0(void\* arg)

{

int i = 10;

while(i > 0){

sem\_wait(&s0);

printf("\nChef 0 is using Pan B to prepare fry 1 ham.\n"); //in Critical Section

sleep(2); //simulate frying 1 ham in 2 seconds

sem\_post(&s0);

i--;

printf("\nChef 0 finished using Pan B.\n"); //exit Critical Section

sleep(1); //transition taking 1 second

}

}

//thread t1 represents Chef 1

void\* thread\_t1(void\* arg)

{

int j = 10;

while(j > 0){

sem\_wait(&s0);

printf("\nChef 1 is using Pan B to prepare fry 1 egg.\n"); //in Critical Section

sleep(2); //simulate frying 1 egg in 2 seconds

sem\_post(&s0);

j--;

printf("\nChef 1 finished using Pan B.\n"); //exit Critical Section

sleep(1); //transition taking 1 second

}

}

int main()

{

//these values are used to identify thread number

int thread0\_id=0, thread1\_id=1;

sem\_init(&s0, 0, 1);

//intialize two variables type pthread\_t called t0 and t1

pthread\_t t0,t1;

//create two threads, passing thread number as the forth argument

pthread\_create(&t0,NULL,thread\_t0,&thread0\_id);

pthread\_create(&t1,NULL,thread\_t1,&thread1\_id);

//join two threads after they finished running

pthread\_join(t0,NULL);

pthread\_join(t1,NULL);

sem\_destroy(&s0);

return 0;

}

**Appendix B (businessman-solution.c)**

// C program to demonstrate a simple multi-threaded process using two threads.

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <semaphore.h>

sem\_t A; //semaphore variable called A is declared

sem\_t B; //semaphore variable called B is declared

int buf\_A, buf\_B; //declare global variables buf\_A and buf\_B

//thread A does the following

void\* thread\_A(void\* arg)

{

int var\_A;

while(1){

var\_A = 1234;

buf\_A = var\_A; //Jot down own contact number to buf\_A so that thread\_B can

copy

sem\_post(&B); //Here is my card

sem\_wait(&A); //Waiting for yours

var\_A = buf\_B; //Copy thread\_B's contact number to var\_A

printf("\nthread\_A: var\_A = %d", var\_A); //print var\_A on screen

sleep(1);

}

}

//thread B does the following

void\* thread\_B(void\* arg)

{

int var\_B;

while(1){

var\_B = 4321;

buf\_B = var\_B; //Jot down own contact number to buf\_B so that thread\_A can

copy

sem\_post(&A); //Here is my card

sem\_wait(&B); //Waiting for yours

var\_B = buf\_A; //Copy thread\_A's contact number to var\_B

printf("\nthread\_B: var\_B = %d", var\_B); //print var\_B on screen

sleep(1);

}

}

int main()

{

//these values are used to identify thread number

int threadA\_id=0, threadB\_id=1;

//initialize semaphore A and B values as 0 (3rd argument)

sem\_init(&A, 0, 0);

sem\_init(&B, 0, 0);

//intialize two variables type pthread\_t called t0 and t1

pthread\_t tA,tB;

//create two threads, passing thread number as the forth argument

pthread\_create(&tA,NULL,thread\_A,&threadA\_id);

pthread\_create(&tB,NULL,thread\_B,&threadB\_id);

//join two threads after they finished running

pthread\_join(tA,NULL);

pthread\_join(tB,NULL);

//print the value of x and y after threads finished running

printf("The final value of buf\_A is %d\n", buf\_A);

printf("The final value of buf\_B is %d\n", buf\_B);

//destroy the semaphore after finished using it

sem\_destroy(&A);

sem\_destroy(&B);

return 0;

}