**Assignment 3 Report**

**How exactly synchronization is achieved using semaphore in our assignment?**

Two semaphores are used – produced and consumed which works as follows –

1. Produced and consumed semaphores are initialized to ‘0’ and ‘1’ respectively.
2. Now, both producer and consumer start executing on separated threads independently.
3. When producer runs for first time, it puts wait on consumer, decrementing the consumed semaphore to ‘0’.
4. After producer has produced a value, it signals consumer by incrementing produced semaphore to ‘1’.
5. Producer again goes to produce a value and puts a wait on consumed .Since, the consumed is set to ‘-1’. It would suspend its execution.
6. When consumer runs for first time, it puts wait on ‘produced’ semaphore and sets it to 0.Since, there are no produced values, consumer would suspend execution.
7. Once the produced semaphore is set to ‘1’ by step 4, consumer would resume execution.
8. Now, consumer will signal consumed semaphore to increment its value to ‘0’.
9. Again, producer who has been waiting on ‘consumed’ semaphore, would resume execution and produce next values for ‘n’.
10. In a parallel thread, consumer resumes execution as soon as ‘produced’ semaphore is set to ‘1’. It consumes the produced value.
11. Step 8 would be repeated

And the process goes on.

**Can the above synchronization be achieved with just one semaphore? Why or why not?**

**Source Code Changes:**

**prodcons.h**

#include <xinu.h>

#include <stddef.h>

#include <stdio.h>

/\*Global variable for producer consumer\*/

extern int n; /\*this is just declaration\*/

extern sid32 produced, consumed;

/\*function Prototype\*/

void consumer(int count, sid32 consumed, sid32 produced);

void producer(int count, sid32 consumed, sid32 produced);

**produce.c**

#include <prodcons.h>

void producer(int count, sid32 consumed, sid32 produced)

{

//Code to produce values less than equal to count,

int i;

for(i = 1; i <= count; i++)

{

wait(consumed);

n = i;

printf("produced: %d \n",n);

signal(produced);

}

}

**consume.c**

#include <prodcons.h>

void consumer(int count, sid32 consumed, sid32 produced)

{

while (1){

wait(produced);

printf("consumed: %d \n",n);

if ( n == count){

break;

}

signal(consumed);

}

}

**xsh\_prodcons.c**

#include <prodcons.h>

#include <ctype.h>

int n ; //Definition for global variable 'n'

/\*Now global variable n will be on Heap so it is accessible all the processes i.e. consume and produce\*/

sid32 produced,consumed;

shellcmd xsh\_prodcons(int nargs, char \*args[])

{

//Argument verifications and validations

int count = 2000; //local varible to hold count

int i = 0;

consumed = semcreate(1);

produced = semcreate(0);

// Initialise the value of n to 0, since this is an extern variable, it may start with the previous value

/\* Output info for '--help' argument \*/

if (nargs == 2 && strncmp(args[1], "--help", 7) == 0)

{

printf("Usage: %s\n\n", args[0]);

printf("Description:\n");

printf("\tProducer Consumer Example using semaphore synchronization.\n");

printf("Options (one per invocation):\n");

printf("\t--help\tdisplay this help and exit\n");

return 0;

}

/\* Check argument count \*/

/\* If argument count is greater than 2, then there are too many arguments\*/

if (nargs > 2)

{

fprintf(stderr, "%s: too many arguments\n", args[0]);

return 1;

}

/\* If argument count is equal to 2, then assign args[1] to count variable \*/

if (nargs == 2)

{

// Parse through the array of parameters and return 1 if there is a character other than a number.

for(i = 0; args[1][i] != '\0'; i++ )

{

if (isdigit(args[1][i]) == 0)

{

fprintf(stderr, "%s: input parameter should be an integer.\n", args[0]);

return 1;

}

}

// Else, it can be safely converted to a number.

count = atoi(args[1]);

}

if(count == 0){

fprintf(stderr, "Count should be greater than zero.\n");

return 1;

}

//create the process producer and consumer and put them in ready queue.

//Look at the definitions of function create and resume in exinu/system folder for reference.

resume( create(producer, 1024, 20, "producer", 3, count, consumed, produced) );

resume( create(consumer, 1024, 20, "consumer", 3, count, consumed, produced) );

}

**Function Descriptions:**

* 1. resume():
  2. create(function, size, priority, name, args, hhkhkkj):

The create system call is used to create a new process that will execute instructions written in the ‘function’ specified in the first argument.

Following is the argument description -

* size specifies the stack size, generally in bytes.
* Priority specifies the priority of the process.
* Name specifies identifying name for the new process.
* args specifies the number of arguments required for ‘function’.
* hhkhkkj specifies the actual parameter that is passed to the process. i.e. count
* This function returns the pid of the created process.
* The created process is in the suspended state.
* The resume function accepts the pid of the process and resumes the execution of the process.
* void producer(int count)
  1. void producer(int count, sid32 consumed, sid32 produced)
* This is the first method passed to the create system call.
* The producer method accepts 3 arguments- count, consumed semaphore and produced semaphore. It puts a wait on consumed sempahore, assigns incremental values to n starting from 1 with step count as 1 and then signals produced semaphore.
  1. void consumer(int count, sid32 consumed, sid32 produced)
* This is the second method that is passed to the system call.
* The consumer method accepts 3 arguments – count, consumed semaphore and produced semaphore. It puts a wait on produced semaphore, prints values of ‘n’ until it reaches its maximum limit i.e. count and signals consumed semaphore.
  1. semcreate(int semValue):

**Contributions –**

**Akshay Kamath (akkamath)**

**Sameedha Bairagi (sbairagi)**