Lab 2

Tolaz Hewa

500590503

CPS590-01

**Lab 2: Producer Consumer with Semaphores**

The goal of this problem is to solve producer/consumer problem using semaphores. You will use the pthreads package to create P producer threads and C consumer threads. Each producer thread inserts NITEMS of integer numbers into a circular buffer of size N. Each consumer thread removes NITEMS items from the buffer. Each thread then repeats the process. Your program should accepts input for P, C and N. The NITEMS is a fixed number both for producer and consumer. Use POSIX semaphore (sem\_init, sem\_get, sem\_wait, sem\_post). The pseudo-code is given in Fig.1. Note that the pseudocode only suggests a guideline. Feel free to add more parameters, variables, and functions as they become necessary. Initialize all necessary variable in a separate function.

• Show buffer indices, items, thread number as a result

• Show the result without and with semaphore variables.

• Is there any problem without semaphore variable?

• Can semaphore variables solve the problem?

**Submission instructions:**

* Show your code with result in the lab.
* Submit a single pdf file on the drop box in D2L.
* File name: Lab2-student number.pdf
* You should put your name, student number on the file.
* Include the lab question also
* Then put your C program
* Sample input to the program
* Sample output

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define NITEMS 3

typedef struct circ\_buff\_tag {

int length;

int \*base;

int \*head;

int \*tail;

}circ\_buff;

int P, C, N;

circ\_buff buffer;

sem\_t stock\_sem;

sem\_t free\_sem;

pthread\_mutex\_t lock\_mut;

pthread\_t \*cT;

pthread\_t \*pT;

void \*consumer(void \*arg);

void \*producer(void \*arg);

void circ\_buff\_insert(circ\_buff \*cb, int item);

int circ\_buff\_remove(circ\_buff \*cb);

void circ\_buff\_print(circ\_buff \*cb);

int circ\_buff\_init(circ\_buff \*cb, int l);

void \*consumerWithout(void \*arg);

void \*producerWithout(void \*arg);

void circ\_buff\_free(circ\_buff \*cb);

int main(int argc, char \*argv[])

{

int i;

char \*buff = (char\*)calloc(sizeof(char), 24);

printf("Please enter the number of producer threads: ");

fgets(buff, 24, stdin);

P = (int)strtol(buff, NULL, 10);

printf("Please enter the number of consumer threads: ");

fgets(buff, 24, stdin);

C = (int)strtol(buff, NULL, 10);

printf("Please enter size of buffer: ");

fgets(buff, 24, stdin);

N = (int)strtol(buff, NULL, 10);

cT = (pthread\_t\*)calloc(sizeof(pthread\_t), C);

pT = (pthread\_t\*)calloc(sizeof(pthread\_t), P);

circ\_buff\_init(&buffer, N);

int \*arr = (int\*)calloc(sizeof(int), P+C);

sem\_init(&stock\_sem, 0, 0);

sem\_init(&free\_sem, 0, N);

printf("\n\nWith Mutual Exclusions\n\n\n\n");

if (pthread\_mutex\_init(&lock\_mut, NULL) != 0)

{

printf("\n mutex init failed\n");

return 1;

}

for(i = 0; i < P; i++)

{

arr[i] = i + 1;

pthread\_create(&(pT[i]), NULL, producer, &arr[i]);

}

for(; i < C + P; i++)

{

arr[i] = i + 1;

pthread\_create(&(cT[i - P]), NULL, consumer, &arr[i]);

}

for(i = 0; i < P; i++)

{

pthread\_join(pT[i], NULL);

}

for(i = 0; i < C; i++)

{

pthread\_join(cT[i], NULL);

}

printf("\n\nWithout Mutual Exclusions\n\n\n");

circ\_buff\_free(&buffer);

circ\_buff\_init(&buffer, N);

for(i = 0; i < P; i++)

{

arr[i] = i + 1;

pthread\_create(&(pT[i]), NULL, producerWithout, &arr[i]);

}

for(; i < C + P; i++)

{

arr[i] = i + 1;

pthread\_create(&(cT[i]), NULL, consumerWithout, &arr[i]);

}

for(i = 0; i < P; i++)

{

pthread\_join(pT[i], NULL);

}

for(i = 0; i < C; i++)

{

pthread\_join(cT[i], NULL);

}

sem\_destroy(&free\_sem);

sem\_destroy(&stock\_sem);

pthread\_mutex\_destroy(&lock\_mut);

free(arr);

free(pT);

free(cT);

circ\_buff\_free(&buffer);

printf("Finished\n");

return 0;

}

void circ\_buff\_insert(circ\_buff \*cb, int item)

{

\*(cb->head) = item;

(cb->head)++;

if(cb->head >= cb->base + cb->length)

{

cb->head = cb->base;

}

}

void circ\_buff\_free(circ\_buff \*cb)

{

free(cb->base);

}

int circ\_buff\_remove(circ\_buff \*cb)

{

int item = \*(cb->tail);

\*(cb->tail) = 0;

(cb->tail)++;

if(cb->tail >= cb->base + cb->length)

{

cb->tail = cb->base;

}

return item;

}

void circ\_buff\_print(circ\_buff \*cb)

{

int i;

printf("Buffer: [");

for(i = 0; i < cb->length-1; i++)

{

printf("%d,", cb->base[i]);

}

printf("%d]\n\n", cb->base[cb->length - 1]);

}

int circ\_buff\_init(circ\_buff \*cb, int l)

{

if((cb->base = (int\*)calloc(sizeof(int), l)) == 0)

{

printf("Circular Buffer memory allocation unsuccessful!\n");

return -1;

};

cb->length = l;

cb->head = cb->base;

cb->tail = cb->base;

return 0;

}

void \*producer(void \*arg)

{

int i;

for(i = 0; i < NITEMS; i++)

{

int item = (rand() % 20) + 1;

sem\_wait(&free\_sem);

pthread\_mutex\_lock(&lock\_mut);

printf("Produce: item: %d, index: %lu, thread: %d [P]\n", item, (buffer.head - buffer.base), \*(int\*)arg);

circ\_buff\_insert(&buffer, item);

circ\_buff\_print(&buffer);

pthread\_mutex\_unlock(&lock\_mut);

sem\_post(&stock\_sem);

}

return NULL;

}

void \*consumer(void \*arg)

{

int i;

for(i = 0; i < NITEMS; i++)

{

sem\_wait(&stock\_sem);

pthread\_mutex\_lock(&lock\_mut);

printf("Consume: item: %d, index: %lu, thread: %d [C]\n", \*(buffer.tail), buffer.tail - buffer.base, \*(int\*)arg);

circ\_buff\_remove(&buffer);

circ\_buff\_print(&buffer);

pthread\_mutex\_unlock(&lock\_mut);

sem\_post(&free\_sem);

}

return NULL;

}

void \*producerWithout(void \*arg)

{

int i;

for(i = 0; i < NITEMS; i++)

{

int item = (rand() % 20) + 1;

printf("Produce: item: %d, index: %lu, thread: %d [P]\n\n", item, (buffer.head - buffer.base), \*(int\*)arg);

circ\_buff\_insert(&buffer, item);

}

return NULL;

}

void \*consumerWithout(void \*arg)

{

int i;

for(i = 0; i < NITEMS; i++)

{

printf("Consume: item: %d, index: %lu, thread: %d [C]\n\n", \*(buffer.tail), buffer.tail - buffer.base, \*(int\*)arg);

circ\_buff\_remove(&buffer);

}

return NULL;

}

**SAMPLE INPUT/OUTPUT:**

Tolazs-MacBook-Pro:lab2 tolaz$ ./a

Please enter the number of producer threads: 3

Please enter the number of consumer threads: 3

Please enter size of buffer: 12

With Mutual Exclusions

Produce: item: 8, index: 0, thread: 1 [P]

Buffer: [8,0,0,0,0,0,0,0,0,0,0,0]

Produce: item: 10, index: 1, thread: 2 [P]

Buffer: [8,10,0,0,0,0,0,0,0,0,0,0]

Produce: item: 14, index: 2, thread: 3 [P]

Buffer: [8,10,14,0,0,0,0,0,0,0,0,0]

Produce: item: 19, index: 3, thread: 1 [P]

Buffer: [8,10,14,19,0,0,0,0,0,0,0,0]

Consume: item: 8, index: 0, thread: 4 [C]

Buffer: [0,10,14,19,0,0,0,0,0,0,0,0]

Consume: item: 10, index: 1, thread: 5 [C]

Buffer: [0,0,14,19,0,0,0,0,0,0,0,0]

Consume: item: 14, index: 2, thread: 6 [C]

Buffer: [0,0,0,19,0,0,0,0,0,0,0,0]

Produce: item: 11, index: 4, thread: 2 [P]

Buffer: [0,0,0,19,11,0,0,0,0,0,0,0]

Produce: item: 13, index: 5, thread: 3 [P]

Buffer: [0,0,0,19,11,13,0,0,0,0,0,0]

Produce: item: 5, index: 6, thread: 1 [P]

Buffer: [0,0,0,19,11,13,5,0,0,0,0,0]

Consume: item: 19, index: 3, thread: 4 [C]

Buffer: [0,0,0,0,11,13,5,0,0,0,0,0]

Consume: item: 11, index: 4, thread: 5 [C]

Buffer: [0,0,0,0,0,13,5,0,0,0,0,0]

Consume: item: 13, index: 5, thread: 6 [C]

Buffer: [0,0,0,0,0,0,5,0,0,0,0,0]

Produce: item: 19, index: 7, thread: 2 [P]

Buffer: [0,0,0,0,0,0,5,19,0,0,0,0]

Produce: item: 4, index: 8, thread: 3 [P]

Buffer: [0,0,0,0,0,0,5,19,4,0,0,0]

Consume: item: 5, index: 6, thread: 4 [C]

Buffer: [0,0,0,0,0,0,0,19,4,0,0,0]

Consume: item: 19, index: 7, thread: 5 [C]

Buffer: [0,0,0,0,0,0,0,0,4,0,0,0]

Consume: item: 4, index: 8, thread: 6 [C]

Buffer: [0,0,0,0,0,0,0,0,0,0,0,0]

Without Mutual Exclusions

Produce: item: 10, index: 0, thread: 1 [P]

Produce: item: 1, index: 1, thread: 1 [P]

Produce: item: 6, index: 1, thread: 2 [P]

Produce: item: 13, index: 2, thread: 1 [P]

Produce: item: 3, index: 2, thread: 3 [P]

Produce: item: 8, index: 3, thread: 2 [P]

Consume: item: 10, index: 0, thread: 4 [C]

Consume: item: 10, index: 0, thread: 5 [C]

Consume: item: 10, index: 0, thread: 6 [C]

Produce: item: 4, index: 5, thread: 3 [P]

Produce: item: 8, index: 6, thread: 2 [P]

Consume: item: 1, index: 1, thread: 4 [C]

Consume: item: 6, index: 2, thread: 5 [C]

Consume: item: 13, index: 3, thread: 6 [C]

Produce: item: 10, index: 7, thread: 3 [P]

Consume: item: 3, index: 4, thread: 4 [C]

Consume: item: 8, index: 5, thread: 5 [C]

Consume: item: 4, index: 6, thread: 6 [C]

Finished

Tolazs-MacBook-Pro:lab2 tolaz$