**Scenario 1: More Consumers than Producers**

**Graphical user interface, text

Description automatically generated**

**Scenario 2: Equal number of Producers and Consumers**

**Text

Description automatically generated**

**Scenario 3: More Producers than Consumers**

**Graphical user interface, text

Description automatically generated**

**Source Code**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#define TRUE 1

#define FALSE 0

typedef int buffer\_item;

#define BUFFER\_SIZE 8

char FIRST\_INITIAL = 'm';

char \*LAST\_NAME = "anderson";

buffer\_item START\_NUMBER;

typedef struct

{

int index;

} parameter;

buffer\_item produce\_value;

buffer\_item buffer[BUFFER\_SIZE];

pthread\_mutex\_t mutex;

pthread\_mutex\_t value\_mutex;

sem\_t empty;

sem\_t full;

int insertPointer = 0, removePointer = 0;

void \*producer(void \*param);

void \*consumer(void \*param);

int insert\_item(buffer\_item item, int index);

int remove\_item(buffer\_item \*item, int index);

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

{

int sleepTime, producerThreads, consumerThreads;

int i, j;

if (argc != 5)

{

printf("Useage: <sleep time> <producer threads> <consumer threads> <start number>\n");

return -1;

}

sleepTime = atoi(argv[1]);

producerThreads = atoi(argv[2]);

consumerThreads = atoi(argv[3]);

START\_NUMBER = atoi(argv[4]);

/\* Initialize the synchronization tools \*/

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

{

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

return 1;

}

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

{

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

return 1;

}

sem\_init(&empty, 1, BUFFER\_SIZE);

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

produce\_value = START\_NUMBER;

/\* Create the producer and consumer threads \*/

pthread\_t producers[producerThreads];

for (int i = 0; i < producerThreads; i++)

{

parameter \*data = (parameter \*)malloc(sizeof(parameter));

data->index = i;

pthread\_create(&producers[i], NULL, producer, data);

}

pthread\_t consumers[consumerThreads];

for (int i = 0; i < consumerThreads; i++)

{

parameter \*data = (parameter \*)malloc(sizeof(parameter));

data->index = i;

pthread\_create(&consumers[i], NULL, consumer, data);

}

/\* Sleep for user specified number of seconds \*/

sleep(sleepTime);

sem\_destroy(&empty);

sem\_destroy(&full);

if (pthread\_mutex\_lock(&mutex) != 0)

{

return -1;

}

printf("DONE\n");

return 0;

}

void \*producer(void \*param)

{

/\* Implementation of the producer thread -- refer to Figure 5.26 on page 256 \*/

parameter \*data = (parameter \*)param;

int index = data->index;

free(data);

buffer\_item item;

while (TRUE)

{

/\* sleep for a random period of time \*/

sleep(rand() % 5 + 1);

/\*

generate/retrieve item

ensures no duplicates

\*/

pthread\_mutex\_lock(&value\_mutex);

item = produce\_value++;

pthread\_mutex\_unlock(&value\_mutex);

/\* insert item \*/

insert\_item(item, index);

}

}

void \*consumer(void \*param)

{

/\* Implementation of the consumer thread -- refer to Figure 5.26 on page 256 \*/

parameter \*data = (parameter \*)param;

int index = data->index;

free(data);

buffer\_item item;

while (TRUE)

{

/\* sleep for a random period of time \*/

sleep(rand() % 5 + 1);

remove\_item(&item, index);

/\* process item \*/

}

}

int insert\_item(buffer\_item item, int index)

{

/\*

insert item into buffer

return 0 if successful

return -1 if not successful

\*/

if (sem\_wait(&empty) != 0)

{

return -1;

}

if (pthread\_mutex\_lock(&mutex) != 0)

{

return -1;

}

// sem\_wait(&empty);

// pthread\_mutex\_lock(&mutex);

buffer[insertPointer] = item;

insertPointer++;

insertPointer %= BUFFER\_SIZE;

printf("Producer %c%s\_P%d produced %d\n",

FIRST\_INITIAL, LAST\_NAME, index, item);

if (pthread\_mutex\_unlock(&mutex) != 0)

{

return -1;

}

if (sem\_post(&full) != 0)

{

return -1;

}

// pthread\_mutex\_unlock(&mutex);

// sem\_post(&full);

return 0;

}

int remove\_item(buffer\_item \*item, int index)

{

/\*

remove object from buffer

place it in item

return 0 if successful

return -1 if not successful

\*/

if (sem\_wait(&full) != 0)

{

return -1;

}

if (pthread\_mutex\_lock(&mutex) != 0)

{

return -1;

}

// sem\_wait(&full);

// pthread\_mutex\_lock(&mutex);

\*item = buffer[removePointer];

removePointer++;

removePointer %= BUFFER\_SIZE;

printf("Consumer %c%s\_C%d comsumed %d\n",

FIRST\_INITIAL, LAST\_NAME, index, \*item);

if (pthread\_mutex\_unlock(&mutex) != 0)

{

return -1;

}

if (sem\_post(&empty) != 0)

{

return -1;

}

// pthread\_mutex\_unlock(&mutex);

// sem\_post(&empty);

return 0;

}