**Simulation of a Producer-Consumer Solution**

**Project Guideline**

*sem\_init(),sem\_getvalue(),sem\_destroy()-deprecated on Mac OS X,not working*

*thus we use Grand Central Dispatch's dispatch\_semaphore\_t for semaphore operations.*

#include<dispatch/dispatch.h>

dispatch\_semaphore\_t semaphore;

semaphore = dispatch\_semaphore\_create(1); //init value with 1

dispatch\_semaphore\_wait(semaphore,DISPATCH\_TIME\_FOREVER);

…..

dispatch\_semaphore\_signal(semaphore);

dispatch\_release(semaphore); //destroy

**Producer**(per thread):

Number:m

ID range:1~m

Functions:

**●**Creates a random integer in the range 1~10000,print the pair(ID,Created Number)

**●**Writes pair into shared file “input-numbers”

●Places pair in the buffer

●Waits for 1millisecond

●Repeat the above steps

\*\*\*Each producer creates q numbers(one at a time)

\*\*\*After placing all numbers in the buffer,the producer terminates.

**Consumer**(per thread):

Number:n

ID Range:1000001~1000000+n

Functions:

●Remove pair(ID,Created Number) from buffer

●Check if pair number is prime number

T:Print the triple(Producer ID,Consumer ID,number) to the shared file “prime-number”

F:Print the triple(Producer ID,Consumer ID,number) to the shared file “non-prime-numbers”

\*\*\*Each consumer should terminate when all producers have terminated and the buffer is empty

\*\*\*You are allowed to use as many shared and local variables across all the threads.

**Buffer capacity:**

p

**Main Thread**:

●Read the value of m,n,p and q from command line

●Check each element was indeed processed exactly once(By sequence number?)

Documents:

dataStructures.h

producerFunctions.h

consumerFunctions.h

main.cpp

producer.cpp

consumer.cpp

shared files:

Input-numbers

Prime-numbers

Non-prime-numbers

**Necessary Elements**

2 processes,1 for producer,one for consumer.

*Data Structures:*

itemProduced{

int producerID;

int createdNumber;

};

itemConsumed{

int producerID;

int consumerID;

int consumedNumber;

}

*Input From Command Line:*

numToProduce //Amount of number for each produce to produce,q

bufferCapacity //Capacity of buffer,p

numProducer //Number of producer

numConsumer //Number of consumer

*Semaphores:*

//semInputDoc didn`t use finally

//semPrimeDoc didn`t use finally

//semNonPrimeDoc didn`t use finally

semBuffer

*Bounder Buffer:*

itemBuffer

**Pesudocode:**

int main(){

main\_terminateSignal = 0;

parameters\_read from command line:

numProducer,numConsumer, numToProduce, bufferCapacity;

create\_producerThread(numProducer);

create\_consumerThread(numConsumer);

pthread\_join() wait for all producerThread terminate;

wait(!main\_terminateSignal);

main\_thread terminate;

}

=========================================================================

create\_producerThread:

create numProducer of producer\_thread;

for each producer\_thread{

create q random numbers;

for each random number{

make itemProduced(produceID,createdNum); //node

sem\_wait(semBuffer);

detect if itemBuffer is full:

if not:

push into itemBuffer;

write into Input-numbers

else:

drop itemProduced;

sem\_signal(semInputDoc);

}

}

=========================================================================

create\_consumerThread:

create numConsumer of consumer\_thread;

for each consumer\_thread{

\*\*\*=> sem\_wait(semBuffer);

detect if itemBuffer is empty:

if not empty:

consume one item;

\*\*\*=> sem\_signal(semBuffer);

make itemConsumed(producerID,consumerID,consumedNum);

detectPrime(item.number);

}

=========================================================================

detectPrime(item.number):

if number is prime:

sem\_wait(semPrimeDoc);

write into Prime-numbers

sem\_signal(semPrimeDoc);

else:

sem\_wait(semNonPrimeDoc);

write into Non-prime-numbers

sem\_signal(semNonPrimeDoc);

=========================================================================

//Ignore these code below

int main(){

pid\_t producer,consumer,pid;

if ((producer = fork()) < 0)

{

perror("fork");

abort();

}

else if ( producer == 0) {

producerRunning();

exit(0);

}

if(producer != 0){

if( (consumer = fork()) < 0 ){

perror(“fork”);

abort();

}

else if( consumer == 0 ){

consumerRunning();

}

}

waitForTermination();

}

//unknown code

/\* Wait for children to exit. \*/

int status;

pid\_t pid;

while (n > 0) {

pid = wait(&status);

printf("Child with PID %ld exited with status 0x%x.\n", (long)pid, status);

--n;

// TODO(pts): Remove pid from the pids array.

}

}