Production Line Simulation
2018-2019

Author: Nemanja Nedic
A.M. 1115201400124

HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837

Contents

Intro

Compile & Run

Makefile is provided for compilation.
Other: make clean, make count

Before running make sure shm segment keys 0x1111 and 0x2222 and 0x3333 are available. If not a bash script that cleans your IPCs for you is provided. Make sure its executable (chmod +x **./cleanIPC.sh**). You should run it just to be safe.

Run:

```
./prodline int num_of_products
```

Organization of directories and files

Inside each file you will find the following:

main.cpp: General outline of the program(Init Shared memory, Set random times, etc) and fork()'s that create the other processes.

ErrorCodes.hpp: error codes used to handle irregular termination

Component.h: struct Component, struct Product. Also this is where you can #define the max times for each operation(manufacture,paint,check,assemble).

SharedMem.h: struct SHMemQueue and its operations along with its semaphore operations

ManufactureStage.h: Code for the process that manufactures Components.

PaintStage.h: Code for the process that paints Components

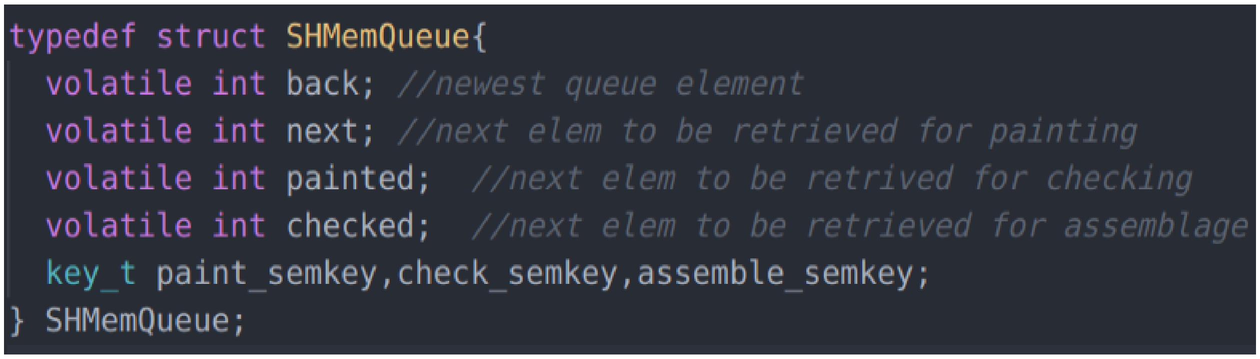
CheckStage.h: Code for the process that checks Components

AssemblageStage.h: Code for the process that assembles Components and produces Products.

Design Outline

Shared Memory

When a Component is manufactured it is placed in a Queue. This is the SharedMemoryQueue (or SHMemQueue). Three of these Queues are crated, one for each type of Component.



```
typedef struct SHMemQueue{
  volatile int back; //newest queue element
  volatile int next; //next elem to be retrieved for painting
  volatile int painted;  //next elem to be retrived for checking
  volatile int checked;  //next elem to be retrieved for assemblage
  key_t paint_semkey,check_semkey,assemble_semkey;
} SHMemQueue;
```

Whenever a Shared Memory Segment is created a SHMemQueue is initalized at its start, That means that the first Component is at sizeof(SHMemQUeue) offset from the starting address of the segment.
Therefore the allocated size of a SHM Segment is:
sizeof(SHMemQueue)+num_component*sizeof(Component). 3 of these are allocated, one per Component type.

All processes have access to at least one SHMemQueue based on the type of Component they are operating on, except Paintshop and Assemblage process which have access to all three SHMemQueue's.
It holds info such as which is the next element to be painted or checked. Processes use this info to retrieve Components and operate on them. Semaphores (discussed below) control weather a process can retrieve a Component from the Queue.

A SHMemQueue is initialized once in main.cpp (parent process), attached and deatched in every process once, and in the end destroyed in main.cpp again. The parent process ofcourse waits for all children to terminate with a wait() loop before destroying SHMemQueue's and terminating.

Queue Semaphores

In order to control access to the Queue, System V IPC semaphores were used as taught in class.

A process, for example the assemblage, cannot retrieve a Component of type1 if it hasn't yet been checked. Therefore a semaphore that controls the number of Components of type1 that are waiting to be assembled is set. When this semaphore's value is 0 and the Paintshop tries to SemDown() it will be blocked, until a Component is checked and ready to be Assembled.

All processes are based on this logic. In conclusion 9semaphores exist in total, 3 per Queue. This could also be implemented as a semaphore set, but i decided against it because there is the advantage of being able to destroy each semaphore on its own. This can prove useful in case a process is blocked on a semaphore and that is "upped" by a process that has already terminated and destroyed the semaphore. Being able to destroy that single semaphore before the process terminates prevents the other process begin blocked by SemDown(). The errno would be set to EIDRM for the programmer to handle.

Each process does SemGet()'s to get the relevant semaphores. The semaphores are destroyed when Assemblage is finished.

Semaphore keys are set upon QueueInit() using:
ftok("./include/SharedMem.h",(int)shkey+i)
Where shkey is the shared memory key_t (hardcoded in main.cpp) and i is the queue identifier (0,1 or 2). Therefore every process with access to a SHMemQueues, can get its semaphores since it has the keys.

Component ids

These are not generated randomly. The 1st of 4 digits of a comonent id specified the component type (1,2 or 3). The rest are the incrementing order in which the component was created. The Product id is a concat of these.

Bottleneck resolution

There is an obvious bottleneck in the porduction line at the Paintshop. There is only 1 Paintshop that handles 3 different types of Components. A adequate policy must be implemented so that CheckStage idle time is minimized:

Paint the component type with the min number of components available for the checking stage. But ignore queues whose paint_semaphores are down, you don't have any components to pull from them anyway. If all semaphores are down then pick a type whose parts have not been painted yet.

Average Paint & Assemblage times

With the following set in Component.h:
#define MAX_MANUFACTURE_TIME 100 //in miliseconds
#define MAX_PAINT_TIME 10
#define MAX_CHECK_TIME 10
#define MAX_ASSEMBLAGE_TIME 10

The subsequent times are generated randomly evry time the program is run.

time\num_components	500	1000	2000
Avg Paintshop Wait Time	0.444769	0.194738	1.033579
Avg Assemblage Wait Time	0.787018	0.345838	1.992308

The End