e-Build
On-Demand Manufacturing System
PA02: Concurrent Processes & IPC

Description

In this project (see Figure 1), we focus on the Manufacturing state of the *On-Demand Manufacturing System* described in PA-01. We will implement only the Manufacturing state as a **Parent** process that creates ***N*** child processes representing concurrent **Factory Lines** in order to fulfill the manufacturing of items in a given customer order. An additional child process, called the "**Supervisor**", will monitor the progress of the factory line processes, collect statistics, and alert the parent (via a rendezvous semaphore) when all factory lines are done. Then, the Parent shall instruct the Supervisor to print the final manufacturing report.

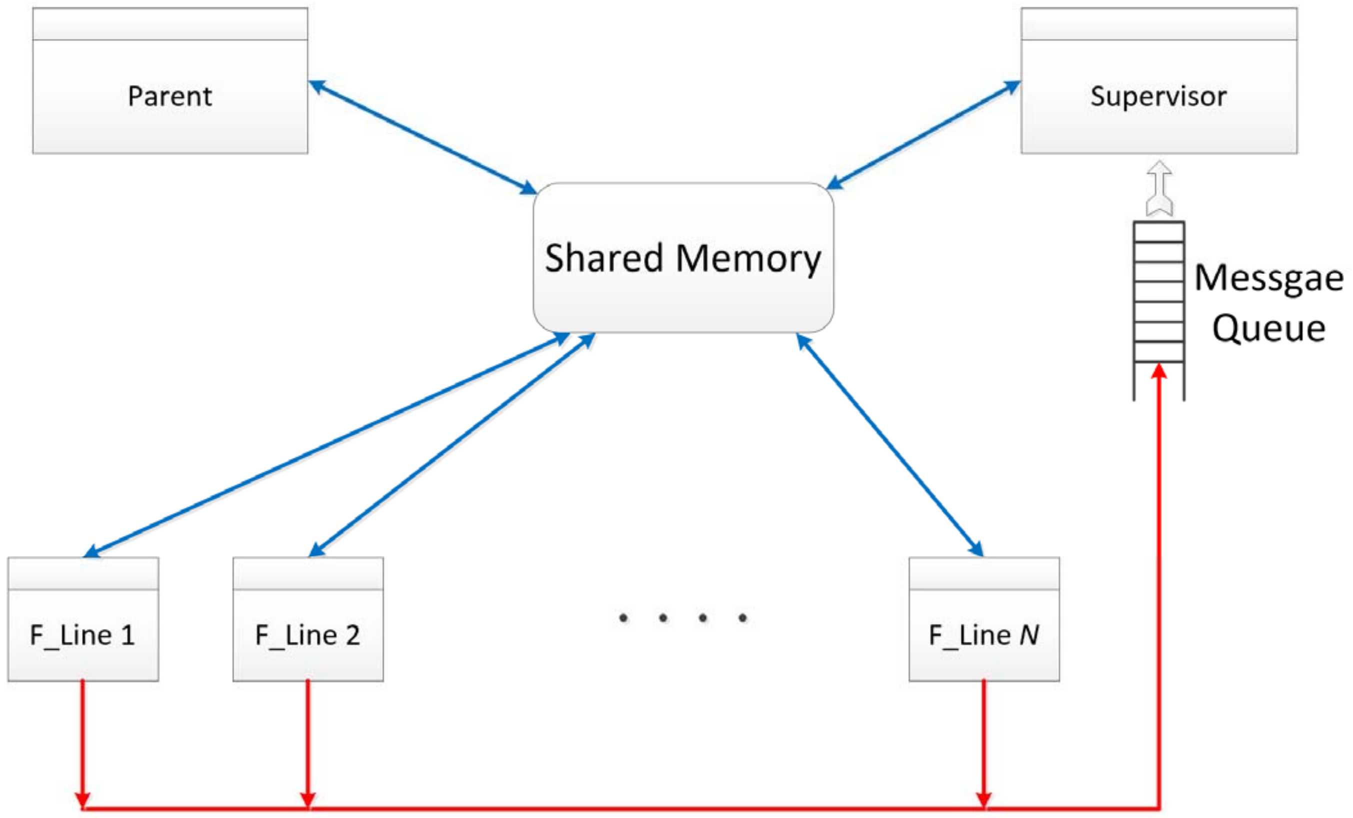


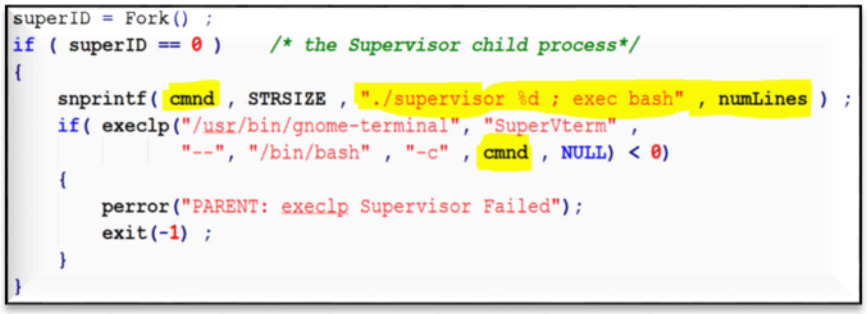
Figure 1 System Architecture

The Parent Process

The parent process starts by creating a shared memory segment for communication amongst all N+2 processes[1], as well as a message queue used by the factory lines to report activity to the Supervisor. The Parent receives the desired number of factory lines ***N*** as its **argv[1]**, and the **order_size** (i.e. total number of parts ordered) as its **argv[2]**. It also sets up/initializes any necessary synchronization mechanisms.

The shared memory contains data such as: the order size, the #of parts made so far, the #of parts still to be made, etc.

Next, the parent, through a series of **fork()** system calls, will launch ***N*** child processes {**FLine1**, **FLine2**, … , **FLine *N*** } representing each of the factory lines, and a last child process "**Supervisor**" in charge of collecting then reporting the achievements of each factory line. The Supervisor process must be executed by the Parent in a separate terminal using code similar to the code shown in Figure 2. In contrast, all factory line processes run in the same terminal as the parent.



```
superID = Fork() ;
if ( superID == 0 )     /* the Supervisor child process*/
{
    snprintf( cmnd , STRSIZE , "./supervisor %d ; exec bash" , numLines ) ;
    if( execlp("/usr/bin/gnome-terminal", "SuperVterm" ,
                "--", "/bin/bash" , "-c" , cmnd , NULL) < 0)
    {
        perror("PARENT: execlp Supervisor Failed");
        exit(-1) ;
    }
}
```

Figure 2 Executing the Supervisor Child

After launching all ***N***+1 processes, the Parent waits for the Supervisor (via a rendezvous semaphore) to know when all factory lines are done making parts.

Next, the parent informs the Supervisor (via a rendezvous semaphore) to print the end-of-manufacturing report and waits for an indication from the Supervisor that the latter has finished printing. At this point, all terminated child processes must be individually waited on for clean up purposes. Also, the shared memory and the message queue should be destroyed to prevent memory leaks.

The pseudo code for the Parent process is shown in Figure 3.

```
{
    Set up shared memory & initialize its objects ;
    Set up the message queue ;
    Fork/Execute Supervisor process to run in a separate terminal ;
    Create/Execute all Factory Line processes (in same terminal as parent) ;
    Wait for Supervisor to collect aggregates from all Factory Lines ;
    Print("Supervisor says all lines have completed") ;
    Give permission to Supervisor to print final report ;
    Wait for supervisor to finish printing ;
    Clean up after zombie processes (Supervisor + all Factory Lines) ;
    Destroy any objects in shared memory ;
    Destroy the shared memory ;
    Destroy message queue ;
}
```

Figure 3 Parent Process

[1] The parent + the Supervisor + the ***N*** factory lines

The Factory Line Processes

The manufacturing lines will collaboratively, and concurrently, work to manufacture the number of items in the order. Each of the factory line processes (see Figure 4) must be implemented as a separate executable program (sharing the same terminal as the parent process) that expects these command-line arguments:

- its **factory_ID** (a unique integer from 1 to ***N***),
- its **capacity** (the maximum number of parts it can make in one iteration), and
- its **duration** (the time in ***milliseconds*** to make that many parts)

The duration and capacity numbers will be generated randomly by the Parent process within the ranges specified below. After each iteration, the factory line process must send a ***Production Message*** to the **Supervisor** via the mailbox indicating: {its **Factory Line_ID**, its **Capacity**, the **number of parts** it has actually made in the iteration just completed, and the **Duration** it took to make those items}. Next, the factory line process determines whether it needs to make more parts. A factory line process must not make more parts than what is still needed, but whenever possible, it must contribute up to its full **capacity** to the concurrent making of parts. If a factory line has to make less parts than its capacity, it will still take the same duration to complete (i.e. no proration of time).

When there are *no more* parts to make, the factory line sends a ***Completion Message*** to the Supervisor's mailbox to inform the latter that the factory line is done. At this time, the factory line process terminates. The pseudo code for the main loop of a Factory Line process is shown in Figure 4. Notice that all factory line processes print to the same terminal of the parent. This is very hazardous. To prevent interference of output, printing to the **`stdout`** device should be treated as a critical section that requires **mutual exclusion**.

```
#iterations = #partsMadeByMe = 0;
While ( remain  > 0 )  {
    Determine how many to make & Update remain ;
    Print("Factory Line %d: Going to make %d parts in %d milliSecs"   , myID,  data ... ) ;
    sleep ( myDuration ) ;
    Create & Send Production message ;
    Increment #iterations ;
    Update total #of parts made by me;
}
Create & Send Completion message ;
```

Figure 4 The Factory Line Process

The Supervisor Process

The supervisor process is implemented as a separate executable running inside its isolated terminal. It receives the number of factory lines ***N***, as a command line argument from the Parent. It also receives production and completion messages from those ***N*** factory line processes, and synchronizes with the Parent via rendezvous-type named semaphores.

The pseudo-code of the Supervisor's main loop is shown in Figure 5.

```
LinesActive = argv[1] ;
While ( LinesActive  > 0 )  {
    Receive a message from the messgaeQueue ;
    If ( Production Message )
        Print("Factory Line %d produced %d parts in %d milliSecs" , data ... ) ;
        Update per-factory-line production aggregates (num-items-built, num-iterations) ;
    Else if ( Termination Message )
        LinesActive -- ;
        Print("Factory Line %d Terminated" , data ...  ) ;
    else
        discard this unsupported message
}
Inform the Parent that all factory lines have completed;
Wait for permission from the Parent to start printing production aggregates ;
Print per-factory-line production aggregates ;
```

Figure 5 The Supervisor Process

Additional Details:

It must be noted that even though the Supervisor and the Factory Line processes are children of the Parent process, the use of execlp() detaches them from any shared memory or message queues previously set up by the Parent before the fork() system call. These queues and/or shared memory segments must be re-attached / re-connected to by the children who have been execlp()'ed.

Note that:

- Use **srandom**() and **random**() to create random number. srandom() must only be called one time to seed the random number generator. Use **time**(NULL) as the seed.
- **capacity** : a random integer in the range 10 to 50, inclusive
- **duration**: a random integer in the range 500 to 1200 **<u>milliseconds</u>**, inclusive. Use **usleep**() to simulate the elapsed time.

- The **capacity** and **duration** of each manufacturing line **<u>should not</u>** be global variables, and must set prior to creating each process and sent to that process via its command-line arguments.
- Named semaphores must be used to ensure mutual exclusion of shared data and to synchronize activities between the concurrent processes. The semaphores should not prevent concurrency when mutual exclusion is not necessary. You lose big points for committing such sinful programming act.
- Determine the necessary system header files (i.e. <somefile.h>) to include in your source code. (hint: man pages)

Output:

In addition to the output from phase 1:

From the Parent Process	From the Factory Process	From the Supervisor Process
• The value of **order_size** • At time of **fork**-ing, each manufacturing line's **capacity** and **duration**	Inside each Iteration: • #parts & duration After Completed: • Total #parts made • Total #iterations	• Echo Factory Lines' Production msgs Final Report: • Total #iterations and #parts made per Factory Line • Grand total #parts made by all. Verify it matches **order_size**.

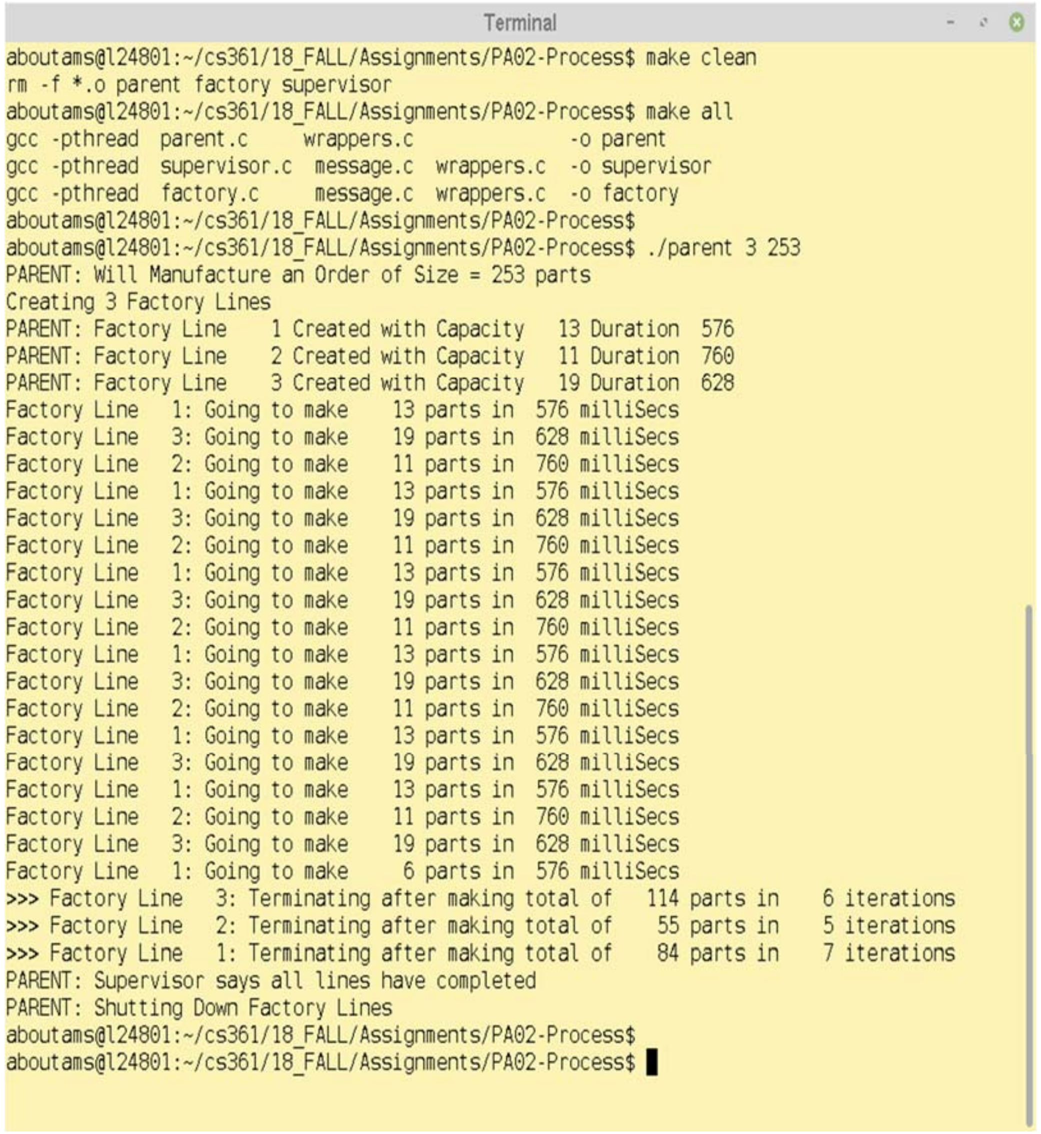
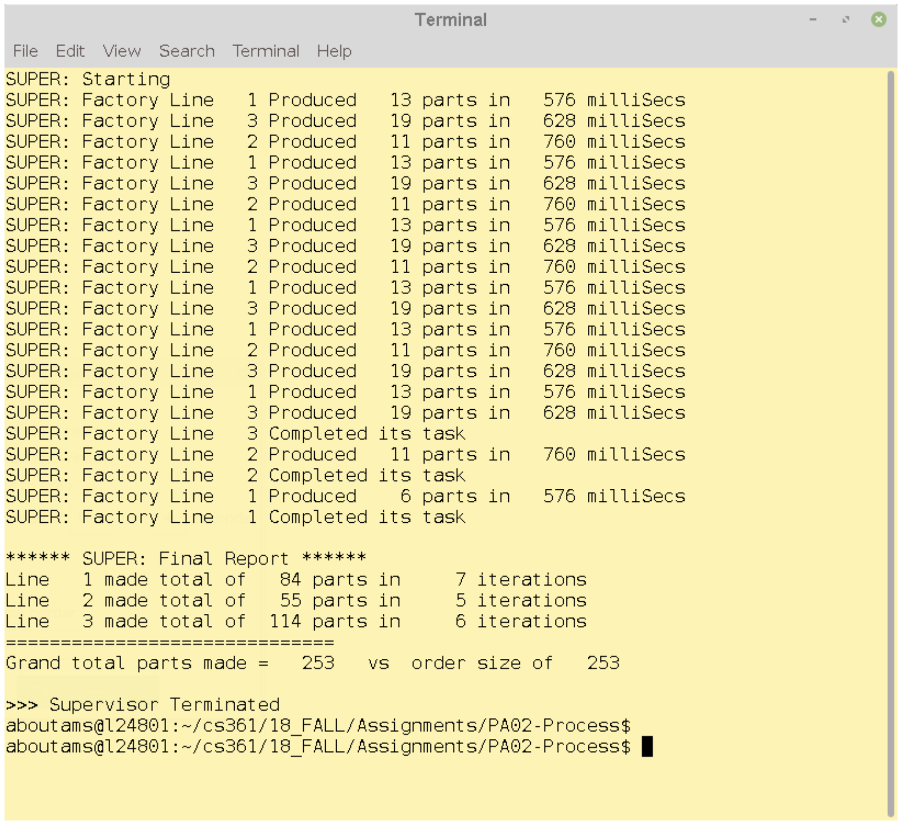
Submission:

Include a **`makefile`** with your source code in a **`.tar`** file named

`pa02-<your last name(s)>.tar`.

Your submissions will only be compiled, linked and executed on **`stu.cs.jmu.edu`** (or a lab computer). Compilation and linking will be done on the command line using the makefile you provide.

Sample Run

Parent & Factory Lines' Terminal	Supervisor's terminal
	

```
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$ make clean
rm -f *.o parent factory supervisor
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$ make all
gcc -pthread  parent.c     wrappers.c             -o parent
gcc -pthread  supervisor.c  message.c  wrappers.c  -o supervisor
gcc -pthread  factory.c     message.c  wrappers.c  -o factory
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$ ./parent 3 253
PARENT: Will Manufacture an Order of Size = 253 parts
Creating 3 Factory Lines
PARENT: Factory Line    1 Created with Capacity   13 Duration  576
PARENT: Factory Line    2 Created with Capacity   11 Duration  760
PARENT: Factory Line    3 Created with Capacity   19 Duration  628
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   2: Going to make    11 parts in  760 milliSecs
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   2: Going to make    11 parts in  760 milliSecs
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   2: Going to make    11 parts in  760 milliSecs
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   2: Going to make    11 parts in  760 milliSecs
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   1: Going to make    13 parts in  576 milliSecs
Factory Line   2: Going to make    11 parts in  760 milliSecs
Factory Line   3: Going to make    19 parts in  628 milliSecs
Factory Line   1: Going to make     6 parts in  576 milliSecs
>>> Factory Line   3: Terminating after making total of   114 parts in    6 iterations
>>> Factory Line   2: Terminating after making total of    55 parts in    5 iterations
>>> Factory Line   1: Terminating after making total of    84 parts in    7 iterations
PARENT: Supervisor says all lines have completed
PARENT: Shutting Down Factory Lines
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$ 
```

```
SUPER: Starting
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   2 Produced   11 parts in   760 milliSecs
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   2 Produced   11 parts in   760 milliSecs
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   2 Produced   11 parts in   760 milliSecs
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   2 Produced   11 parts in   760 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   1 Produced   13 parts in   576 milliSecs
SUPER: Factory Line   3 Produced   19 parts in   628 milliSecs
SUPER: Factory Line   3 Completed its task
SUPER: Factory Line   2 Produced   11 parts in   760 milliSecs
SUPER: Factory Line   2 Completed its task
SUPER: Factory Line   1 Produced    6 parts in   576 milliSecs
SUPER: Factory Line   1 Completed its task

****** SUPER: Final Report ******
Line   1 made total of   84 parts in     7 iterations
Line   2 made total of   55 parts in     5 iterations
Line   3 made total of  114 parts in     6 iterations
==============================
Grand total parts made =   253   vs  order size of   253

>>> Supervisor Terminated
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$
aboutams@l24801:~/cs361/18_FALL/Assignments/PA02-Process$ 
```