

Assignment 2

Control of Pick and Place Machine for SMT Assembly Using Multiple Processes

Date Due: Thursday 25th July 2024

Marks: 350 (35% of total assessment marks)

Penalty for Late Submission: Per the current [USQ Assessment Procedure](#)

Artificial Intelligence (AI) Utilization for this Assessment Item

Level	Requirement
0	No Artificial Intelligence to be Used: Artificial intelligence is not to be used in this assessment item beyond basic editing features (spelling and grammar) in Microsoft Word, Apple Pages or Grammarly and similar tools. No automation of written text allowed. Failure to adhere to this requirement will result in referral to the Academic Integrity Unit (AIU) for investigation.

The Artificial Intelligence Use in Assessment form must be completed to declare your use of AI for this assessment item. A completed form must be submitted for your assessment to be marked.

Academic Misconduct

For the purposes of this assignment, academic misconduct includes (but is not necessarily limited to):

- contract cheating i.e., arranging for a third party to design, implement and/or test your program
- copying the design, implementation and/or report of another student
- collusion with other students to design, implement and/or test your program
- using generative AI tools to generate part or all of your assignment submission

Outline

In this assignment, you will design, implement, test and document a C program which spawns multiple processes during runtime to control and simulate a simple pick and place machine for assembly of Surface Mount Technology (SMT) based Printed Circuits Boards (PCBs). The processes need to co-operate with each other using inter-process communication and synchronization. Some enhancements to the control functionality are required relative to Assignment 1, specifically automatic loading and unloading of PCBs to/from the pick and place machine in autonomous mode. In addition, you will be managing the simulation of the pick and place machine (as one of the co-operating processes) and so the source code for the simulator is provided to you.

There are three deliverables:

1) A text readable and non-compressed/non-zipped Word or PDF **report** documenting your system design, implementation and testing. The report must include the following concise information:

- | | |
|-----------------|---|
| System design | - A brief explanation of significant design choices, particularly with respect to the order of process spawning, inter-process communication and synchronization (maximum 1 page) |
| Testing results | - The test cases you executed and the associated results. These are best represented in tabular form. |
| Program listing | - The commented C source code in an appendix |

2) A Windows **zip file** (i.e. with a native .zip extension, and not produced by 7-Zip or WinRAR or any other incompatible archive program) containing your Code::Blocks project for each process (there will probably be 4 Code::Blocks projects corresponding to 4 co-operating processes) and a Code::Blocks workspace file. See the Annex for more details on the zip file packaging of your solution. Remember to save not only the C source files, but the projects themselves and the workspace file from within Code::Blocks before you build your zip file.

3) The completed and signed Artificial Intelligence Use in Assessment form (available on the StudyDesk) – you must indicate which AI usage level your submission corresponds to.

System Specification

Design

The application must be designed using **multiple processes**. The following 4 processes shall be included (note that the first process spawns the other processes at runtime):

1. A Startup and Monitor process that sets up any common variables (e.g. pipes and/or semaphores), spawns the other processes (including the simulator) using the POSIX `fork()` call in an intelligent order and monitors the other processes for expected or unexpected termination.
 - As each child is spawned, the Startup and Monitor process shall print the name (e.g. “Controller”) and associated pid of the process.
 - When a child of the Startup and Monitor process terminates, the Startup and Monitor process shall print the name and pid of the process which has terminated and the associated termination status code.
 - The child processes spawned at startup shall overlay themselves using the POSIX `exec1()` call, passing any required command line arguments and using relative file paths to executables.
 - As far as possible, all messages which are to be printed by the Startup and Monitor process shall be sent via an anonymous pipe to the Display process for printing. This will not always be possible e.g. before the Display process has been spawned and has overlayed itself, in which case it is acceptable for the Startup and Monitor process to print directly to the standard output.
2. A Controller process to control the pick and place machine. The base requirements for this process are the same as those specified in Assignment 1 Part A and Part B. However:
 - In autonomous mode, the first action of the Controller process shall be to request that the PCB be automatically loaded onto the pick and place machine, and its last action shall be to request that the assembled PCB be automatically unloaded from the pick and place machine. For this purpose, you must design a means for the Controller process to instruct the Simulator process to perform the loading/unloading.
 - Any text to be printed shall not be printed directly to the standard output, rather it shall be relayed to the Display process via an anonymous pipe for printing (remember pipes should be point-to-point and unidirectional; you should not re-use the same pipe for communication between different pairs of processes).
 - Some other source code changes will be required to the Controller process to support synchronization.
3. A Display process to display messages received via anonymous pipes from the other processes to the standard output. The rationale for a separate Display process is that corruption can occur if multiple processes attempt to print to a single terminal window simultaneously or near simultaneously, therefore all display output should be handled by a single process apart from in exceptional situations as discussed in this specification. When the Display process prints a message, it shall be clear which

process is the ultimate source of that message i.e. which process sent the message via a pipe to the Display process for printing. The identity of the source process can be part of the original message itself or can be added to the original message by the Display process. The Display process shall print messages in simulation time order, even those which originate from different source processes (note: the simulation time to be displayed for a message is that recorded by the source process, not the simulation time when the Display process reads the message from a pipe).

4. The Simulator process which simulates the operation of the pick and place machine.
 - The Simulator process shall be capable of loading and unloading the PCB on request from the Controller process. It also shall indicate to the Controller process when the requested loading/unloading is complete. You may assume that each of the loading and unloading actions takes 1.5 seconds. The simulation of loading/unloading simply involves waiting for this time period to elapse before signaling the action complete, and passing appropriate messages to the Display process for printing.
 - Any text to be printed shall not be printed directly to the standard output, rather it shall be relayed to the Display process via an anonymous pipe for printing (remember pipes should be point-to-point and unidirectional; you should not re-use the same pipe for communication between different pairs of processes).
 - Some other source code changes will be required to the Simulator process to support synchronization.

In Assignment 1, you may occasionally have seen unexpected behaviour (e.g. the simulator not executing an instruction requested by the controller or apparently skipping over it), especially in autonomous mode. This is because of race conditions caused by specific scheduling decisions for the controller and simulator made by the operating system. Your synchronization design should eliminate this problem so that unexpected behaviour does not occur irrespective of the exact scheduling decisions made. Selective use of synchronization is important because if you use synchronization where it is not really needed, you may slow down the operation of your program considerably or even create a deadlock between processes.

Explain briefly your synchronization design in your report. You may find it easier or even necessary to use named POSIX semaphores rather than unnamed POSIX semaphores because of the use of process overlaying in the assignment.

Implementation

The source code of the simulator from Assignment 1 will be provided.

You must use only the C standard library and C POSIX library, and the final compiled program must execute correctly on Cygwin as it will only be tested in that environment.

It must be possible to run your program directly from within Code::Blocks by selecting Run (Ctrl-F10) when the Startup and Monitor process is selected as the active project. This is actually simpler than trying to arrange a number of executables to use common resources by manually placing them in certain directories. When you run the Startup and Monitor process from within Code::Blocks, the associated project directory becomes the working directory for all input and output for all processes which are spawned. For example, all processes can refer to the same memory mapped file for shared memory using just the name of the file (i.e. without an associated file path), and that file will be created/updated in the project directory associated with the Startup and Monitor process. Similarly, the “centroid.txt” file need only be placed in the project directory associated with the Startup and Monitor process even though it is the Controller process that reads this file.

An Assignment 1 sample solution will be provided for the controller which you may or may not choose to use as the basis of your Assignment 2 i.e. you can use your own solution or the sample solution.

Special notes on the use of synchronization

Synchronization is a difficult concept, and you may find it very challenging to design, implement and test for it. I recommend you implement synchronization after all your other code is written and basically working, allowing you to fall back to a code version which is functional if the synchronization element cannot be implemented correctly (e.g. if you create a deadlock with your synchronization code in which different processes are waiting for each other and cannot progress).

One of the challenges with synchronization is testing whether it is working correctly or at all. I am not expecting you to write detailed test cases for synchronization, because it is difficult to provoke race conditions that might lead to unexpected program behaviour, and these race conditions might occur very infrequently. However, you should have some general idea and comment on whether your synchronization implementation has had the desired effect:

- If you still occasionally see unexpected program behaviour, the synchronization code is definitely not having the desired effect.
- If you never see unexpected program behaviour, then perhaps the synchronization code is working as designed (but this of course is not guaranteed as a race condition might not have occurred while you were observing).

If you want to observe/confirm the value of a semaphore at any point in a program, there is a POSIX function `sem_getvalue()` for this purpose.

You may occasionally experience a subtle problem with named semaphores during development because of their persistence. For example, assume your program exits abnormally

without first unlinking/deleting the named semaphore when the value of the semaphore is 0. Because named semaphores are persistent unless deleted, the next time you run the program, the semaphore is not re-created with a specific intended value, but instead starts with the value 0 for all processes, which results in each process waiting at the first invocation of `sem_wait()` on that semaphore. Therefore, the program may appear to stall or freeze. You can try to manually delete the relevant named semaphore at `C:\cygwin64\dev\shm` and then restart your program. However, this may only solve the problem temporarily, because if the program exits abnormally again without first unlinking/deleting the named semaphore, the same situation will eventually transpire. Ultimately, you must solve the issue that is causing the program to exit abnormally.

Important note:

Programs submitted must be the work of each individual student. Exchange of ideas on conceptual issues between students is allowed, but students must *write their own programs* and not share code.

Assessment 2 Rubric:

The rubric is presented on the following page. In addition to the rubric, there are also penalties for not complying with this specification in certain areas as follows. These are at the discretion of the examiner.

Issue	Penalty
The packaging of submitted files does not meet the specified requirements (see page 1 and Annex)	Up to 70 marks
When compiling the source files, there are unresolved compiler warnings (e.g. variables being used uninitialized) which can cause the program to fail to work correctly on one machine while working correctly on another.	Up to 35 marks
It is not possible to run the program successfully from within Code::Blocks without making source code changes	Up to 70 marks

ELE4307 Assignment 2 marking criteria

Each attribute below relates to a key element of the assignment, marked on a scale of 0 to 100%. Zero marks will be awarded for no attempt.











Attribute	Poor	Adequate	Good	Excellent	Mark
Functionality	The program completely fails to respond to manual or auto operation as specified (including the cases of compile time or runtime errors)	The program correctly responds to some manual and/or auto operations per the specification	The program correctly responds to most manual and/or auto operations per the specification	The program correctly responds to all manual and/or auto operations per the specification	/ 80
Documentation (e.g. design choices, test plan/results)	Documentation is poorly written or does not explain design/implementation/testing	Documentation is limited or explanation of design/implementation/testing is incomplete	Documentation is reasonably well written and explains design/implementation/testing but contains some flaws.	Documentation is well written and explains design/implementation/testing clearly and logically.	/ 50
Program Modularity	The program is not split into individual processes as required	The program includes a Simulator and Controller process, but not the other required processes.	The program includes most, but not all, required processes.	All mandatory processes have been implemented per the specification	/ 30
Program Implementation	The program is implemented with limited use of accepted programming techniques/comments.	The program is implemented with some use of accepted programming techniques/comments.	The program is implemented with adequate use of accepted programming techniques/comments.	The program is implemented with good use of accepted programming techniques/comments.	/ 30
Implementation of pipes	No or ill judged use of pipes	A reasonable attempt has been made to use pipes per the specification, but the pipes are basically non-functional OR pipes have been re-used between different pairs of processes	The pipes are functional and operate according to the specification, but there are some minor functional or performance issues with their operation.	The pipes are functional and operate according to the specification with no issues.	/ 80
Implementation of synchronization	No or ill judged use of synchronization, or runtime performance is unacceptable	Some operations that require synchronization have been protected, and runtime performance is adequate	Most operations that require synchronization have been protected, and/or very few operations that do not require synchronization have been protected, and runtime performance is good	All operations that require synchronization have been protected, no operations that do not require synchronization have been protected, and runtime performance is excellent	/ 80

Marker's Comments

TOTAL MARK	/ 350
------------	-------

Annex: Packaging Requirements

As discussed earlier, one of your submissions will be a **zip file** containing your Code::Blocks project for each process and a Code::Blocks workspace file. The workspace file allows you to load all projects simultaneously in Code::Blocks rather than one by one, saving you time. Therefore, the contents of your zip file should look something like below. You do not need to use the same names for your project files or workspace file, but you should use meaningful names.

Name	Status
 Assgn2_2024_Controller	
 Assgn2_2024_Display	
 Assgn2_2024_Simulator	
 Assgn2_2024_Startup	
 Assgn2_2024.workspace	

You can create an appropriate workspace file by first loading all your Assignment 2 projects into Code::Blocks, then selecting 'File > Save workspace as...', and finally saving the file in the top level Code::Blocks directory where your projects are also stored.

If you double click on the created workspace file when Code::Blocks is not open, Code::Blocks should start and load all your Assignment 2 projects automatically.

If you open the workspace file in a text editor, it should look something like the following, clearly showing which projects it links to.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<CodeBlocks_workspace_file>
  <Workspace title="Workspace">
    <Project filename="Assgn2_2024_Controller/Assgn2_2024_Controller.cbp" />
    <Project filename="Assgn2_2024_Display/Assgn2_2024_Display.cbp" />
    <Project filename="Assgn2_2024_Simulator/Assgn2_2024_Simulator.cbp" />
    <Project filename="Assgn2_2024_Startup/Assgn2_2024_Startup.cbp" />
  </Workspace>
</CodeBlocks_workspace_file>
```