Lecture 4 - Message Passing and CSP

Alex Bartol

February 9, 2012

1 Communication

Multi-threaded Programs need a way to communicate between processes. This can be done with Message Passing.

There are two options for a message sender

- Wait for acknowledgement from the recipient (synchronous)
- Proceed without verification (asynchronous)

2 Hoare and Message Passing

1978, Hoare proposed an interface for message passing which provided

- Guarded commands are used to "guard" a statement. The guard is a proposition which must be true before the statement is executed. Also, should the statement begin to execute, one may assume the guard to be true.
- Dynamic thread creation
- Synchronous Message Passing allows for a message to be verified to have sent/received Message Passing can also play the role of syncronization since it provides the tools to wait on a process.
- No communication using global variables So there would be no shared memory across threads. This is important because it allows for the message passing interface to scale into multiple processes and machines without really changing the implementation

3 Philosophers Problem, Deadlock, and Livelock

5 philosophers sitting around a round table with a fork between each of them. After an arbitrary about of time (per philosopher) the philosopher will try to pick up both the utensil on his left and his right in order to eat. This can only be done if neither of the philosophers adjacent to him/her are currently holding the utensil he/she is trying to aquire.

This example applies to computer science often in the form of deadlock or livelock. Deadlock is a state where two or more competing actions are blocked by waiting for mutual resources. Livelock is very similar to deadlock except with livelock, states are constantly changed but there is no progressing. Livelock is a special case of resource starvation. The forks, in the dining philosophers problem, can be thought of as system resources (either physical hardware or data values). Each thread (or philosopher) must aquire two locks before other threads aquires either of them. Also, given that a particular thread can aquire a resource, it should not wait for the other one since there is a circle of dependency.

Hoare's Message Passing API is as follows

- X ? (a,b) ⇒ get a pair of arbitrary variables from process X and store them in local variables 'a' and 'b'
- Y ! (3,n) ⇒ send the value 3 and the local variable 'n' to Y. The variable will be passed by value and not by reference.
- X(i) ? V() ⇒ send the result of executing the function V() to the ith process of X
- display(i-2) ! "A" ⇒ send the character "A" to the (i-2) process of display

4 Example of Hoare's Message Passing API

```
X:: *[c:char, A?c ->
        [ c <> "*" --> B!c
        []
          c = "*" --> A?c;
                        [ c <> "*" --> B!"*"; B!c
                        []
                          c = "*" --> B!"#" ]
        ]  ]
```

Figure 1: Program on Slide 7

4.1 Explaination

This program is reading and sending characters. X gets a char given from A. If the char is a "*", then send the char to B. Then X gets a new char from A. If the new char is a "*" as well, it will send the character "#" to B. Otherwise it will forward the new char to B.

4.2 Assumptions

- A is sending characters to it
- B is listening for characters for it to be sent
- A sends a non-terminating stream

5 Bounded Buffers

Bounded Buffer is a fixed length of storage for data. Two integers are kept to reference the buffer from (say 'r' and 'w', signifying read and write). When a thread reads into the buffer it will read from index 'r' and increment 'r'. When a thread writes into the buffer it will write into index 'w' and increments 'w'. Should 'r' come to equal 'w' then the reader should stop to allow the writer(s) to fill the buffer. Should 'w' approach 'r' then the writers should stop in order to let the reader(s) to catch up. All indexes are mod sizeof(buffer).

Bounded buffers use guarded commands. Since the reader and writer both check the variables 'r' and 'w' there needs to be a form of synchronization to avoid threads overwriting data being accessed by the other thread.

For an example of a Bounded Buffer, see page 8 of the Lecture-4 notes