CSCI 322
Principles of Concurrent Programming
Filip Jagodzinski
Announcements

HW2

• Due tonight

Wednesday, 27 July

• Final Exam
• In-class
• You can bring with you one piece of paper with notes
• No electronic help of any kind is allowed

Thursday, 28 July

• Last lab (Message passing)
Announcements

Lab 2 ... creating an array of threads

```cpp
int numThreads = 5;
std::future<double> *threads = new std::future<double>[numThreads];
for (int i=0; i<numThreads; ++i) {
    threads[i] = std::async(std::launch::async, myFunction, arg1, arg2);
}
```
From last time

Butterfly Barrier: Combines multiple 2-process barriers. At stage $s$, synchronize with a process $2^{s-1}$ away. The number of processes must be a power of 2.
From last time

S1 | ______ | ______ | ______ |
   | ______ | ______ |
   | ______ |

S2 | ______ | ______ |
   | ______ |
   | ______ |

Dissemination barrier
(Last stage not shown)
Use barriers to “double the distance” at which elements are added.

From last time

Q: Advantage/speedup? : After $\log_2 n$ rounds all partial sums have been computed
From last time

Q: What are three shortcomings of semaphores?
Q: What are three shortcomings of semaphores?

Semaphores are a low-level mechanism – it is easy to make errors when using them.

1. You (a programmer) must be careful NOT to omit an increment or decrement somewhere in your code, or to use the wrong semaphore (in case there are multiple ones being used)
2. Semaphores are also global, thus you must examine the entire program to know how they are used
3. Semaphores provide BOTH mutual exclusion and synchronization techniques, but what if we want to use these concepts independently?
From last time

Monitors

• **Active processes** (threads running concurrently) interact by calling procedures in the same monitor
• The monitors are referred to as **passive**
• A monitor is used to group together the representation and implementation of a shared resource.
• In different languages, they are created in different ways

- Only procedure names are visible to “outside” of the monitor
- Monitors may not access variables declared outside of the monitor
- Permanent variables are initialized before any procedures are called

<table>
<thead>
<tr>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>// condition variables</td>
</tr>
<tr>
<td>// procedures</td>
</tr>
</tbody>
</table>
From last time

A monitor is following one of these disciplines, but NOT both.

Signal and continue (SC) : the signaler (process that called signal) continues, and the signaled (process that is awakened) executes at some later time

Signal and wait (SW) : the signaler waits until some later time and the signaled process executes immediately.
From last time

Signal and wait (SW): the signaler waits until some later time and the signaled process executes immediately.

If a process is executing, and it issues signal, what happens next, assuming SW?
If a process is executing, and it issues signal, what happens next, assuming SW?

The signaler “waits” by being placed at the end of the entry queue.
Signal and wait (SW): the signaler waits until some later time and the signaled process executes immediately.

If a process is executing, and it issues signal, what happens next, assuming SW?

The signaled process (that is at the head of cv) executes immediately.
Today

Threads
Processes
Processes and Threads

Up until this point, we’ve been talking about “threads”, in which a heap is a shared memory resource that is dynamically allocated.
Processes and Threads

Up until this point, we’ve been talking about “threads”, in which a heap is a shared memory resource that is dynamically allocated.

Threads “communicate” via shared memory resources (an array, for example)
Up until this point, we’ve been talking about “threads”, in which a heap is a shared memory resource that is dynamically allocated.

Threads “communicate” via shared memory resources (an array, for example)
Up until this point, we’ve been talking about “threads”, in which a heap is a shared memory resource that is dynamically allocated.

Threads “communicate” via shared memory resources (an array, for example) or via a `get()` method that retrieves the result of each thread’s calculation.

A threading approach is well suited for a multi-core, single processor architecture.
Processes and Threads

We’ve also talked about Processes in the context of locks and barriers, and critical regions.

Up until now, we haven’t distinguished between a lightweight process and a heavyweight process.
We’ve also talked about Processes in the context of locks and barriers, and critical regions.

Up until now, we haven’t distinguished between a lightweight process and a heavyweight process.

Each of these threads can be thought of a lightweight process, because it is not a self-contained entire program.

Q: So what is a heavyweight process?
Processes and Threads

(heavyweight process) Process Approach

This is a distributed memory model, and the processes can be run on different CPUs on different racks, machines, etc.

Q: How do such processes communicate?

In Contrast …

These are **heavyweight processes** because each has their own heap AND stack.
Processes and Threads

Process Approach

This is a distributed memory model, and the processes can be run on different CPUs on different racks, machines, etc.

Q: How do such processes communicate?

First, there needs to be a physical connection

Q: Is having a “connection” enough to facilitate communication?

In Contrast…

These are **heavyweight processes** because each has their own heap AND stack.
These are **heavyweight processes** because each has their own heap AND stack.

This is a distributed memory model, and the processes can be run on different CPUs on different racks, machines, etc.

Q: How do such processes communicate?

First, there needs to be a physical connection.

There also needs to be a communication protocol.
Message passing is concerned with sending messages either **asynchronously** or **synchronously** among heavyweight processes on a **distributed memory** architecture.

Task: Be able to explain the difference between synchronous and asynchronous message passing. Be able to discuss the pros and cons of each.
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

If you have 2 processes sending messages back and forth to each other, HOW do they communicate?

Hint: Think of how humans communicate
Q: Is there a protocol?
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

If Process 1 wants to communicate with Process 2, how is that achieved? (if you’ve taken Networks, this communication process should be familiar)
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

P1 wants to communicate with P2, but before it can “send” data to P2, it must first ask “hey are you there.” After the request is sent, P1 suspends itself.
Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

P2 receives a request, and invokes the Receive() method, which sends an acknowledgement message to P1 to the effect of “okay I’m listening.” P2 then suspends itself.
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

Once P1 receives the acknowledge message from P2...
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

P1
- Send()
- Suspend()
- Transfer()

P2
- Receive()
- Suspend()

It sends the message (data) via the network, to P2.
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

Which is received and processed by P2.
Synchronous Message Passing

Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

In this scenario, the `Send()` routine is a **blocking routine**, because after a process sends a message, it self blocks.
Synchronous message passing: when a sender and receiver synchronize HOW they communicate...

Q: What are the disadvantages of synchronous message passing?
Synchronous message passing: when a sender and receiver synchronize HOW they communicate ...

Q: What are the disadvantages of synchronous message passing?

1. Concurrency is vastly reduced, because there is always one process that is blocked and waiting.
Synchronous message passing: when a sender and receiver synchronize **HOW** they communicate...

**Q:** What are the disadvantages of synchronous message passing?

1. Concurrency is vastly reduced, because there is always one process that is blocked and waiting.
2. There is the possibility of deadlock when two processes are “coded” incorrect.

**P1**
- Send via ch2
- Receive via ch1

**P2**
- Send via ch1
- Receive via ch2

**Q:** Why will these processes deadlock?

**Q:** How might you fix the deadlock?
Q: What are the disadvantages of synchronous message passing?

1. Concurrency is vastly reduced, because there is always one process that is blocked and waiting.
2. There is the possibility of deadlock when two processes are “coded” incorrect.
3. The method of making the connection is error prone.
Message Passing

Q: How does asynchronous message passing differ from synchronous message passing?
Asynchronous Message Passing

Q: What is the human communication “analogy” to asynchronous message passing?

Hint: Think of how humans communicate in a large group setting
Q: Is there a protocol?
Q: Do people wait until it is their turn to speak?
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.
Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.
Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the disadvantages of an asynchronous broadcast?
Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the disadvantages of an asynchronous broadcast?

1. Flooding of the network, which might cause congestion
Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the disadvantages of an asynchronous broadcast?

1. Flooding of the network, which might cause congestion
2. Communication overhead diminishes a program’s performance, thus sending too many messages is detrimental to a program’s performance
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the disadvantages of an asynchronous broadcast?

1. Flooding of the network, which might cause congestion
2. Communication overhead diminishes a program’s performance, thus sending too many messages is detrimental to a program’s performance
3. P1 has an excessively high computational burden (it makes and sends all the messages)
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

The “human” analogy: When a manager wants to delegate tasks to subordinates, it ...
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

The “human” analogy: When a manager wants to delegate tasks to subordinates, it ...
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using scatter?
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using scatter?

1. Flooding of the network, which might cause congestion if $n$ is large
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using scatter?

1. Flooding of the network, which might cause congestion if $n$ is large
2. P1 must create $n-1$ unique “messages,” whereas in broadcast P1 makes 1 message and sends it out multiple copies
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Once portions of the problem have been sent out (and computations performed), what is the manager’s role?
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Collect the “work done”
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using `gather`?
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using gather?

1. P1 is now the location of a major bottleneck, because it receives upwards of $n-1$ messages
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using gather?

1. P1 is now the location of a major bottleneck, because it receives upwards of $n-1$ messages
2. If each of P2 through Pn finish “at the same time,” network congestion will be large
Asynchronous Message Passing

Humans do not always wait their turn to talk. Some just start talking. Processes can do the same.

Q: What are the drawbacks of using gather?

1. P1 is now the location of a major bottleneck, because it receives upwards of n-1 messages
2. If each of P2 through Pn finish “at the same time,” network congestion will be large
3. Assuming P1 receives all of the messages from P2 through Pn, it must synthesize (combine) them into a coherent whole
Q: Regardless whether broadcast, scatter, or gather are used, their ultimate efficiency is based on what 2 factors?
Q: Regardless whether broadcast, scatter, or gather are used, their ultimate efficiency is based on what 2 factors?

   a) The nature of the problem
   b) The Topology of the network
Scenario: There are 5 processes, each of which has a locally scoped variable. The goal is for EVERY process to learn of the smallest and largest of the locally scoped variables among the 5 processes.
Topologies

Scenario: There are 5 processes, each of which has a locally scoped variable. The goal is for EVERY process to learn of the smallest and largest of the locally scoped variables among the 5 processes.

P1

P2

P3

P5

Q: What kind of topology is best?
Q: Which CPU(s) should perform the calculations?

(on the board discussion)
In a centralized approach, there is a single CPU (1 in this case) that is designated as the coordinator.
In a centralized approach, there is a single CPU (1 in this case) that is designated as the coordinator.

* To guarantee that the largest “local” variable among all 5 processes is known by 5 processes

Fewest Number of messages sent

(On the board discussion)
In a centralized approach, there is a single CPU (1 in this case) that is designated as the coordinator.
Dissemination Approach

In a symmetric approach, all processes (nodes) send and receive message from all others.
Topologies

<table>
<thead>
<tr>
<th>Dissemination Approach</th>
<th>Fewest Number of messages sent</th>
<th>Pros/Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td></td>
<td>(on the board discussion)</td>
</tr>
<tr>
<td>Symmetric</td>
<td></td>
<td>(In class exercise)</td>
</tr>
</tbody>
</table>

In a symmetric approach, all processes (nodes) send and receive message from all others.
Topologies

<table>
<thead>
<tr>
<th>Dissemination Approach</th>
<th>Fewest Number of messages sent</th>
<th>Pros/Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>(on the board discussion)</td>
<td></td>
</tr>
<tr>
<td>Symmetric</td>
<td>(In class exercise)</td>
<td></td>
</tr>
<tr>
<td>Shopping Cart</td>
<td>(In class exercise)</td>
<td></td>
</tr>
</tbody>
</table>
Up Next

MPI
Rendezvous
Final Exam Prep