Processes

Eric Wu (ericwu@cs)

Topics for Today

- Project 1
- Processes
- Parallelism and Concurrency
- Permissions and Privileges

Project 1

Things to consider and remind you:

- NT functions are made from user mode, but are only executed in the kernel
- You need to monitor both the number of times NT functions are called and returned.
  - These may not necessarily be the same!

Project 1

- Questions?
- Concerns?

Processes

Recap from lecture

- What is a process?

Processes

Recap from lecture

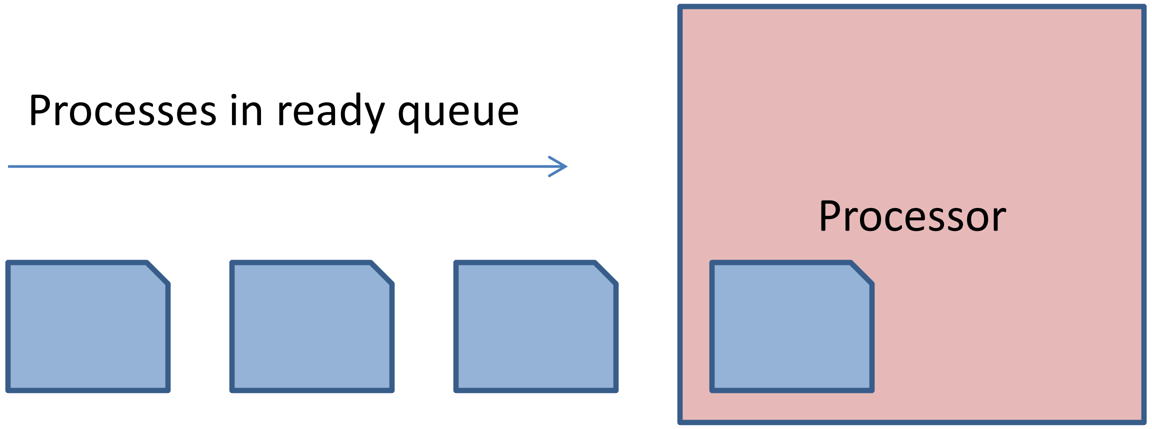
- What is a process?
  - An execution entity
  - A running instance of a program

Processes

How does an OS on a single processor hardware run multiple processes?

Processes

How does an OS on a single processor hardware run multiple processes?



Processes

How about multiple processors?

- The answer depends on implementation…
  - One master processor
  - Master processor schedules processes to itself and secondary processors

Processes

What does the OS do when there are no processes to run?

Processes

What does the OS do when there are no processes to run?

- Run an idle process!
  - Periodically checks for any new tasks to run
  - Loops the HLT instruction to save CPU time

Parallelism and Concurrency

- Executing multiple tasks simultaneously
- This will be a focus for Project 2!

Parallelism and Concurrency

- Executing multiple tasks simultaneously
- This will be a focus for Project 2!

Parallelism and Concurrency

```
NTSTATUS
NTReadFile(...) {
   ...
   CSE451Info.readcalls++;
   return status;
}
```

This is broken on a multiprocessor. Why?

Parallelism and Concurrency

```
NTSTATUS
NTReadFile(...) {
  ...
  int tmp = CSE451Info.readcalls;
  CSE451Info.readcalls = tmp+1;
  return status;
}
```

This is broken on a multiprocessor. Why?

Very subtle...

Parallelism and Concurrency

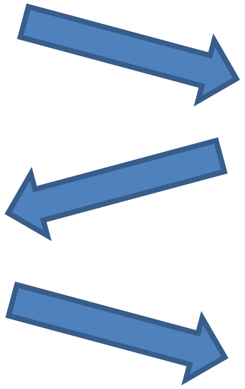
Thread 1: read “hi.txt”

```
//cse451Info.readcalls == 4

NTSTATUS
NTReadFile(...) {
  ...
  int tmp = CSE451Info.readcalls;



  CSE451Info.readcalls = tmp+1;
  return status;
}
```



Thread 2: read “foo.c”

```
//cse451Info.readcalls == 4

NTSTATUS
NTReadFile(...) {
  ...

 int tmp = CSE451Info.readcalls;
  CSE451Info.readcalls = tmp+1;




  return status;
}
```

Parallelism and Concurrency

Thread 1: read “hi.txt”

```
//cse451Info.readcalls == 4

NTSTATUS
NTReadFile(...) {
   ...
   int tmp = CSE451Info.readcalls;



   CSE451Info.readcalls = tmp+1;
   return status;
}


// CSE451Info.readcalls == 5
```

Should be CSE451Info.readcalls == 6!

Thread 2: read “foo.c”

```
//cse451Info.readcalls == 4

NTSTATUS
NTReadFile(...) {
   ...

  int tmp = CSE451Info.readcalls;
   CSE451Info.readcalls = tmp+1;




   return status;
}

// CSE451Info.readcalls == 5
```

Parallelism and Concurrency

How do we solve this consistency issue?

- Use a mutex
  - Denote start (lock) and end (unlock) of a critical section.
  - Ensures critical section only gets accessed by one thread at a time.

Parallelism and Concurrency

Thread 1: read “hi.txt”

```
NTSTATUS
NTReadFile(...) {
  ...
  acquire(&some_mutex);
  int tmp = CSE451Info.readcalls;
  CSE451Info.readcalls = tmp+1;
  release(&some_mutex);
  …
}
```

Thread 2: read “foo.c”

```
NTSTATUS
NTReadFile(...) {
   ...
  acquire(&some_mutex);
  int tmp = CSE451Info.readcalls;
  CSE451Info.readcalls = tmp+1;
  release(&some_mutex);
…
}
```

Protection and Privileges

- All hardware resources must be protected.
- User access to these resources must be restricted.
- Recap: How does the OS issue protection?

Protection and Privileges

Things to consider:

- Why can only the OS create processes?

Protection and Privileges

Things to consider:

- Why can only the OS create processes?
- Can the OS ever deny a user program from issuing a system call? If so, when? If not, why?

Protection and Privileges

Things to consider:

- Why can only the OS create processes?
- Can the OS ever deny a user program from issuing a system call? If so, when? If not, why?
- Files can only be accessed by file descriptors, and not addresses. Why?

Protection and Privileges

Practice Problem (homework prep!)

Which of the following are privileged instructions?

1. Set value of the timer.
2. Read the clock.
3. Clear memory.
4. Issue a trap instruction.
5. Switch from user to kernel mode.
6. Access I/O device.