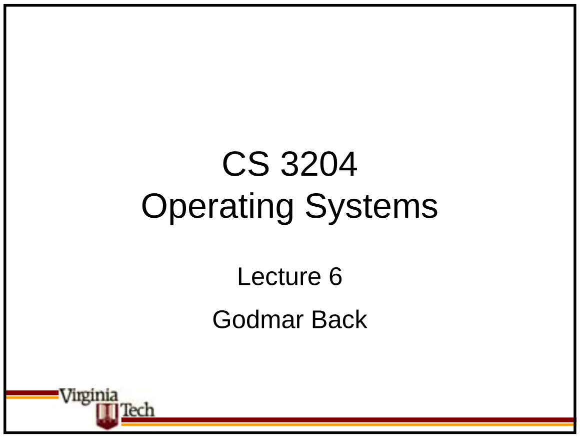
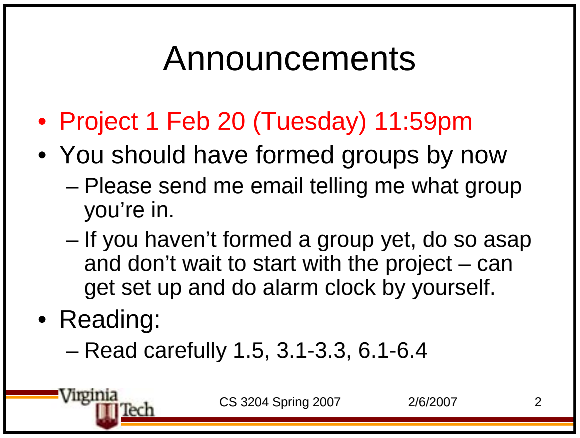


CS 3204
Operating Systems

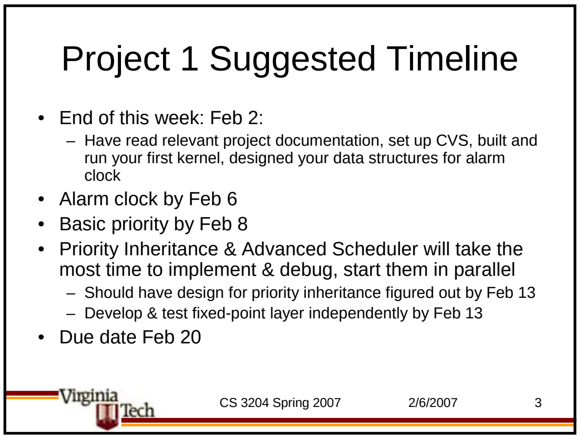
Lecture 6

Godmar Back



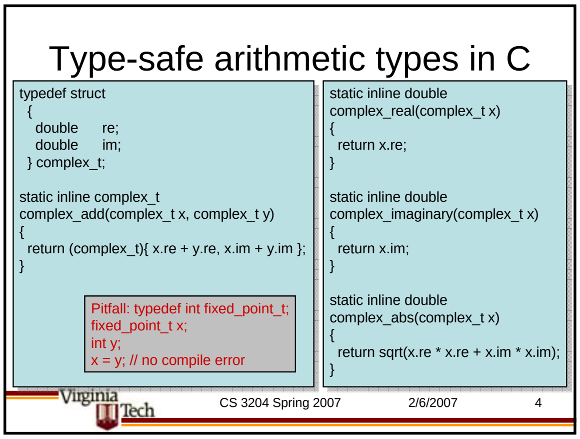
Announcements

- Project 1 Feb 20 (Tuesday) 11:59pm
- You should have formed groups by now
  - Please send me email telling me what group you're in.
  - If you haven't formed a group yet, do so asap and don't wait to start with the project – can get set up and do alarm clock by yourself.
- Reading:
  - Read carefully 1.5, 3.1-3.3, 6.1-6.4



Project 1 Suggested Timeline

- End of this week: Feb 2:
  - Have read relevant project documentation, set up CVS, built and run your first kernel, designed your data structures for alarm clock
- Alarm clock by Feb 6
- Basic priority by Feb 8
- Priority Inheritance & Advanced Scheduler will take the most time to implement & debug, start them in parallel
  - Should have design for priority inheritance figured out by Feb 13
  - Develop & test fixed-point layer independently by Feb 13
- Due date Feb 20

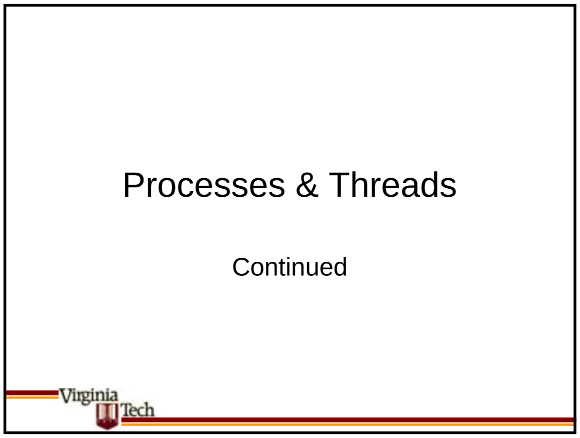


Type-safe arithmetic types in C

```
typedef struct
  {
    double      re;
    double      im;
  } complex_t;

static inline complex_t
complex_add(complex_t x, complex_t y)
{
  return (complex_t){ x.re + y.re, x.im + y.im };
}
```

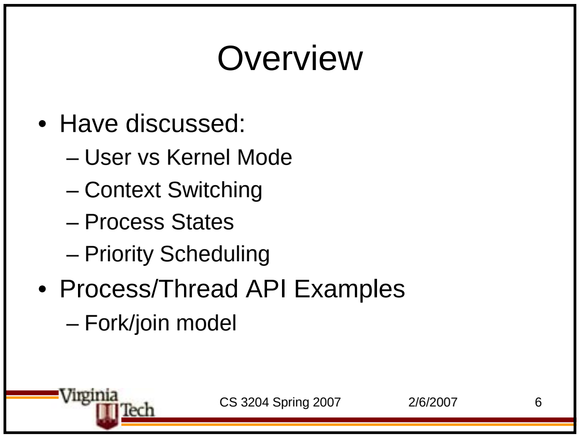
```
Pitfall: typedef int fixed_point_t;
fixed_point_t x;
int y;
x = y; // no compile error
```

```
static inline double
complex_real(complex_t x)
{
  return x.re;
}

static inline double
complex_imaginary(complex_t x)
{
  return x.im;
}

static inline double
complex_abs(complex_t x)
{
  return sqrt(x.re * x.re + x.im * x.im);
}
```



Processes & Threads

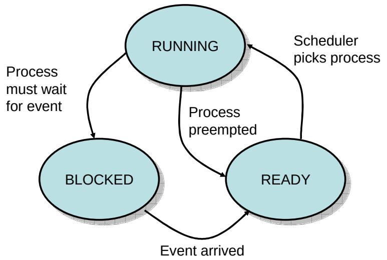
Continued



Overview

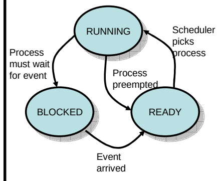
- Have discussed:
  - User vs Kernel Mode
  - Context Switching
  - Process States
  - Priority Scheduling
- Process/Thread API Examples
  - Fork/join model

Process States



- Only 1 process (per CPU) can be in RUNNING state

Using thread_yield() to implement preemption



- Current thread (“RUNNING”) is moved to READY state, added to READY list.
- Then scheduler is invoked. Picks a new READY thread from READY list.
- Case a): there’s only 1 READY thread. Thread is rescheduled right away
- Case b): there are other READY thread(s)
  - b.1) another thread has higher priority – it is scheduled
  - b.2) another thread has same priority – it is scheduled provided the previously running thread was inserted in tail of ready list.
- “thread_yield()” is a call you can use whenever you identify a need to preempt current thread.
- Exception: inside an interrupt handler, use “intr_yield_on_return()” instead

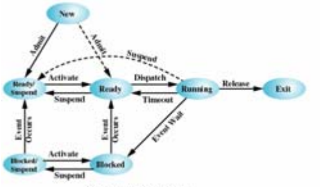
Reasons for Preemption

- Generally two: quantum expired or change in priorities
- Reason #1:
  - A process of higher importance than the one that’s currently running has just become ready
- Reason #2:
  - Time Slice (or Quantum) expired
- Question: what’s good about long vs. short time slices?

I/O Bound vs CPU Bound Procs

- Processes that usually exhaust their quanta are said to be CPU bound
- Processes that frequently block for I/O are said to be I/O bound
- Q.: what are examples of each?
- What policy should a scheduler use to juggle the needs of both?

Process States w/ Suspend

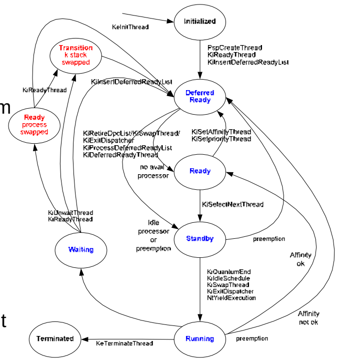


(b) With Two Suspend States

- Can be useful sometimes to suspend processes
  - By user request: ^Z in Linux shell/job control
  - By OS decision: swapping out entire processes (Solaris & Windows do that, Linux doesn’t)

Windows XP

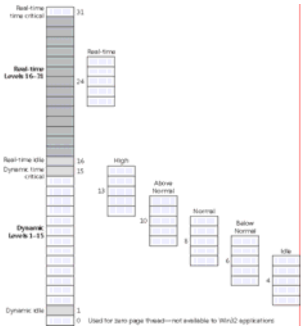
- Thread state diagram in a industrial kernel



- Source: Dave Probert, Windows Internals – Copyright Microsoft 2003

Windows XP

- Priority scheduler uses 32 priorities
- Scheduling class determines range in which priority are adjusted
- Source: Microsoft® Windows® Internals, Fourth Edition: Microsoft Windows Server™



Process Creation

- Two common paradigms:
  - Cloning vs. spawning
- Cloning: (Unix)
  - “fork()” clones current process
  - child process then loads new program
- Spawning: (Windows, Pintos)
  - “exec()” spawns a new process with new program
- Difference is whether creation of new process also involves a change in program

fork()

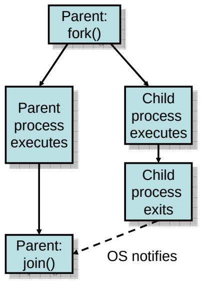
```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int ac, char *av[])
{
    pid_t child = fork();
    if (child < 0)
        perror(“fork”), exit(-1);
    if (child != 0) {
        printf ("I'm the parent %d, my child is  %d\n",
            getpid(), child);
        wait(NULL);    /* wait for child (“join”) */
    } else {
        printf ("I'm the child  %d, my parent is %d\n",
            getpid(), getppid());

        execl("/bin/echo", "echo", "Hello, World", NULL);
    }
}
```

Fork/Exec Model

- Fork():
  - Clone most state of parent, including memory
  - Inherit some state, e.g. file descriptors
  - Important optimization: copy-on-write
    - Some state is copied lazily
  - Keeps program, changes process
- Exec():
  - Overlays current process with new executable
  - Keeps process, changes program
- Advantage: simple, clean
- Disadvantage: does not optimize common case (fork followed by exec of child)

The fork()/join() paradigm

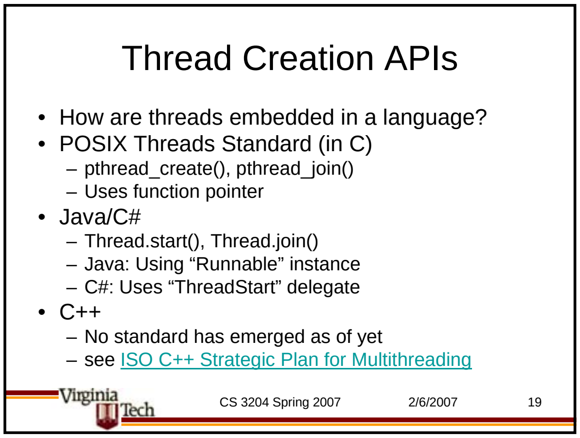
- After fork(), parent & child execute in parallel
- Purpose:
  - Launch activity that can be done in parallel & wait for its completion
  - Or simply: launch another program and wait for its completion (shell does that)
- Pintos:
  - Kernel threads: thread_create (no thread_join)
  - exec(), you’ll do wait() in Project 2



CreateProcess()

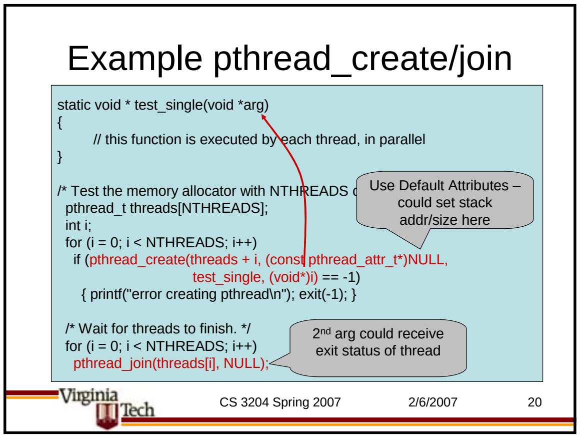
```
// Win32
BOOL CreateProcess(
  LPCTSTR lpApplicationName,
  LPTSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL bInheritHandles,
  DWORD dwCreationFlags,
  LPVOID lpEnvironment,
  LPCTSTR lpCurrentDirectory,
  LPSTARTUPINFO lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation );
```

- See also system(3) on Unix systems
- Pintos exec() is CreateProcess(), not like Unix’s exec()



Thread Creation APIs

- How are threads embedded in a language?
- POSIX Threads Standard (in C)
  - pthread_create(), pthread_join()
  - Uses function pointer
- Java/C#
  - Thread.start(), Thread.join()
  - Java: Using “Runnable” instance
  - C#: Uses “ThreadStart” delegate
- C++
  - No standard has emerged as of yet
  - see ISO C++ Strategic Plan for Multithreading

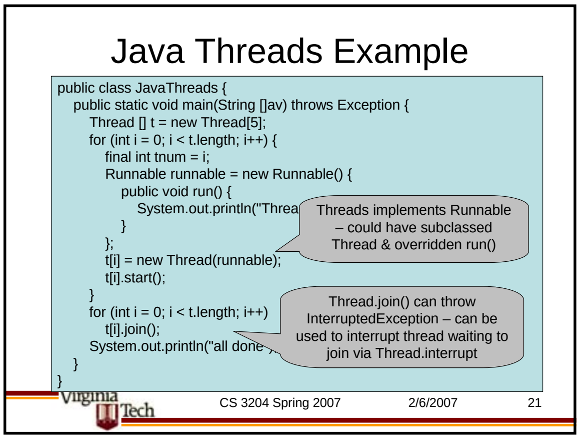


Example pthread_create/join

```
static void * test_single(void *arg)
{
        // this function is executed by each thread, in parallel
}

/* Test the memory allocator with NTHREADS c
  pthread_t threads[NTHREADS];
  int i;
  for (i = 0; i < NTHREADS; i++)
    if (pthread_create(threads + i, (const pthread_attr_t*)NULL,
                                  test_single, (void*)i) == -1)
      { printf("error creating pthread\n"); exit(-1); }

  /* Wait for threads to finish. */
  for (i = 0; i < NTHREADS; i++)
    pthread_join(threads[i], NULL);
```

Use Default Attributes – could set stack addr/size here

2nd arg could receive exit status of thread

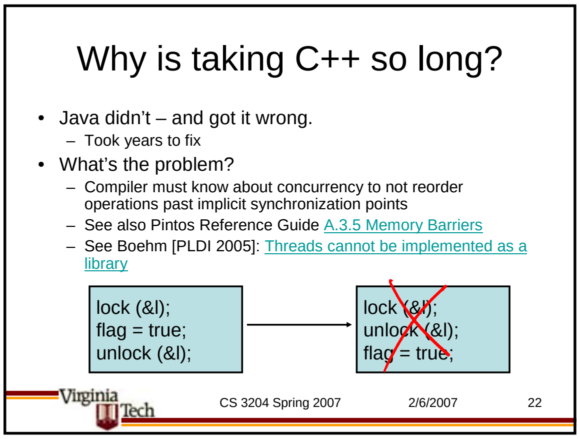


Java Threads Example

```
public class JavaThreads {
    public static void main(String []av) throws Exception {
        Thread [] t = new Thread[5];
        for (int i = 0; i < t.length; i++) {
            final int tnum = i;
            Runnable runnable = new Runnable() {
                public void run() {
                    System.out.println("Threa
                }
            };
            t[i] = new Thread(runnable);
            t[i].start();
        }
        for (int i = 0; i < t.length; i++)
            t[i].join();
        System.out.println("all done
    }
}
```

Threads implements Runnable – could have subclassed Thread & overridden run()

Thread.join() can throw InterruptedException – can be used to interrupt thread waiting to join via Thread.interrupt



Why is taking C++ so long?

- Java didn’t – and got it wrong.
  - Took years to fix
- What’s the problem?
  - Compiler must know about concurrency to not reorder operations past implicit synchronization points
  - See also Pintos Reference Guide A.3.5 Memory Barriers
  - See Boehm [PLDI 2005]: Threads cannot be implemented as a library

lock (&l);
flag = true;
unlock (&l);

~~lock (&l);~~
~~unlock (&l);~~
~~flag = true;~~