CS3204: Operating Systems

Lecture 7: Processes & Threads

Creation, intro to race conditions

Presenter: Ali R. Butt

Announcements

- Project 1 due Monday Feb 23, 11:59pm
- Project 1 help session slides online

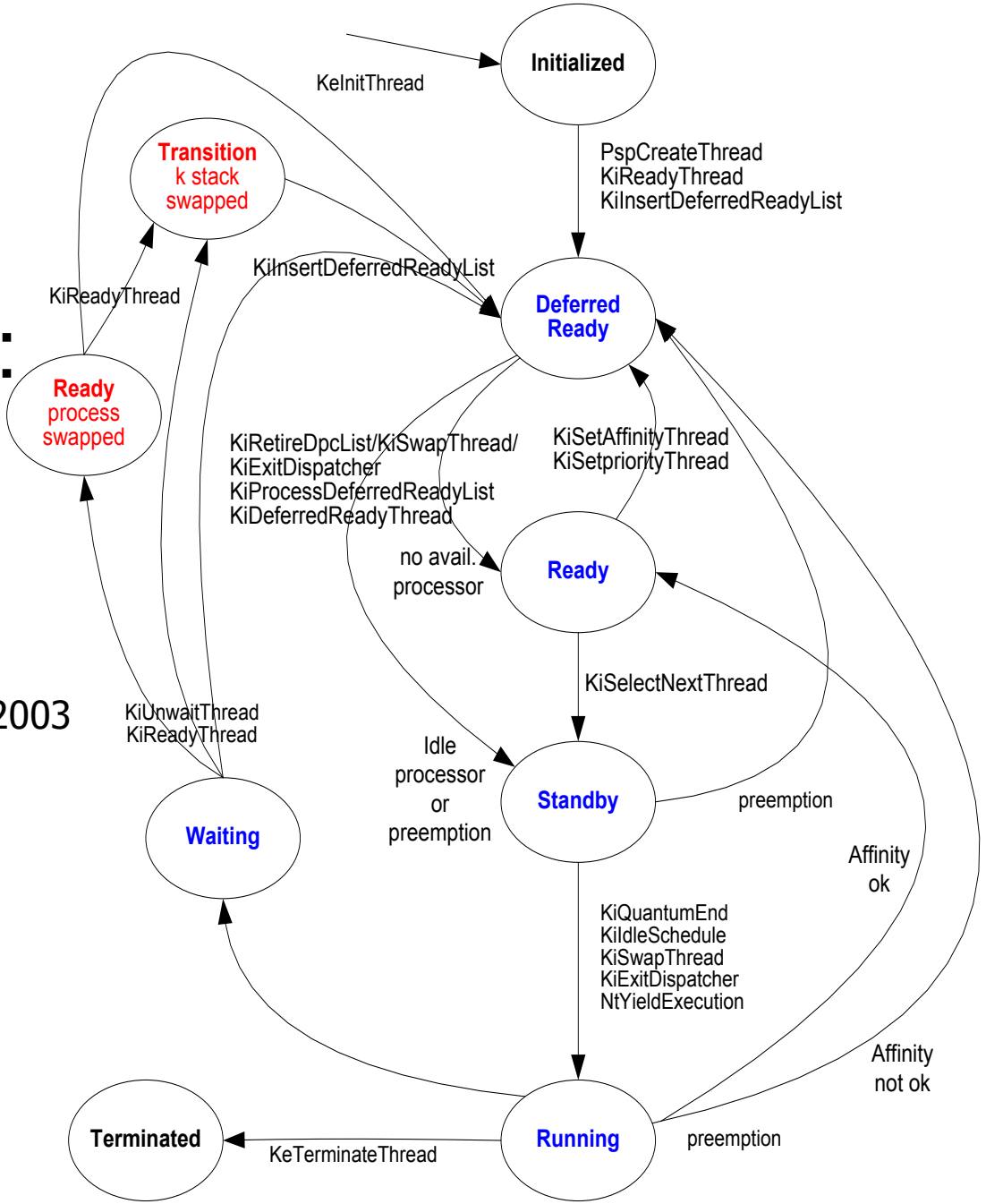
- Reading assignments:
  - ☐ Chapters 1, 2 – skim. Read carefully 1.5.
  - ☐ Read carefully Chapter 3.1-3.3
  - ☐ Read carefully Chapter 6.1-6.4

Project 1 Suggested Timeline

- By now:
  - Have read relevant project documentation, set up CVS, built and run your first kernel, designed your data structures for alarm clock
- Are finishing your Alarm clock
- Basic priority by Feb 13
- 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 15
  - Develop & test fixed-point layer independently by Feb 15
- Due date Feb 23

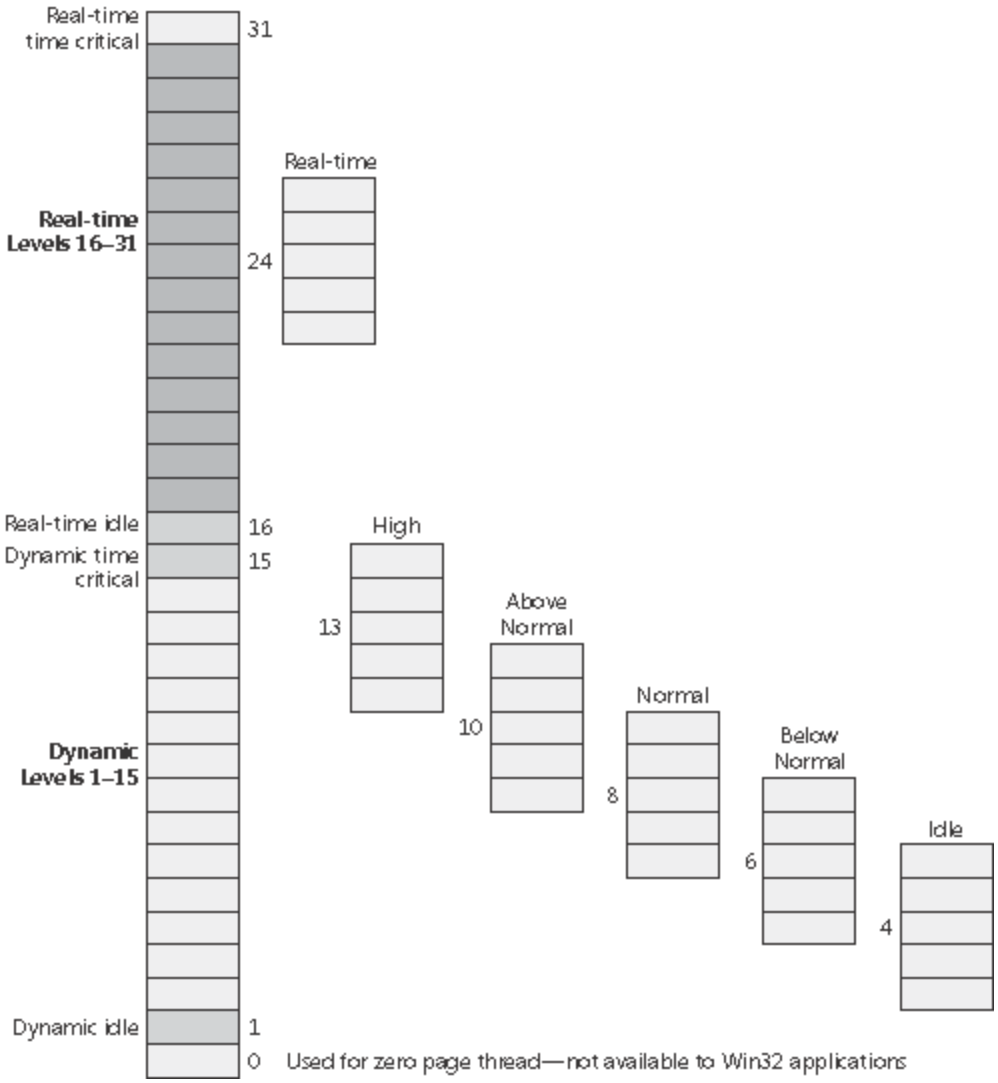
Thread state diagram in an industrial kernel: Windows XP

- 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()

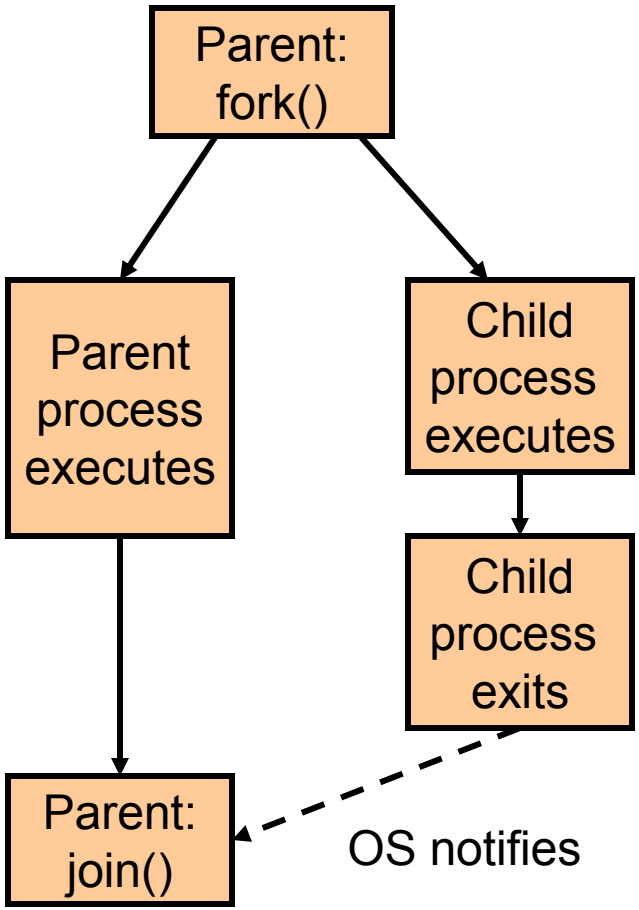
```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

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

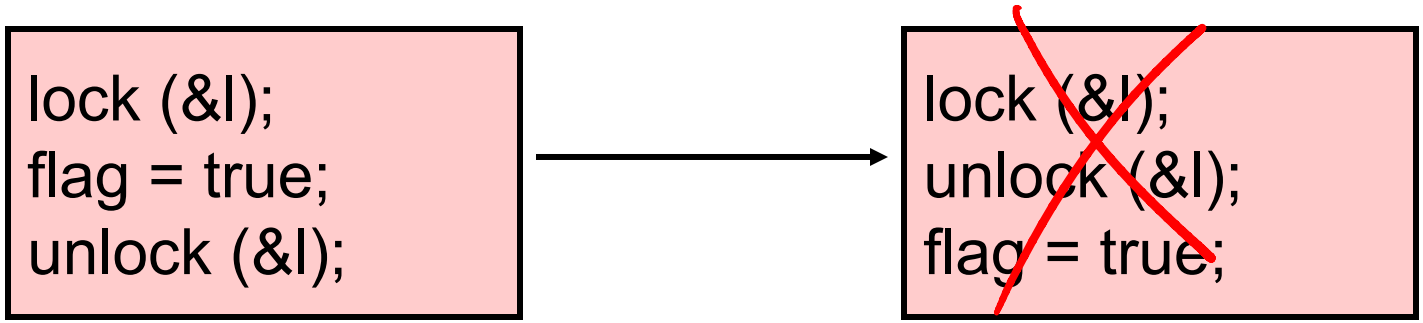
```
public class JavaThreads {
    public static void main(String []av) throws Exception {
        Thread [] t = new Thread[5];
        for (int i = 0; i < t.length; i++) {
            final int tnum = i;
            Runnable runnable = new Runnable() {
                public void run() {
                    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



Processes & Threads (Summary)

- Had looked at APIs with which to create processes/threads
- Spawning vs. cloning
- "fork/join" paradigm (will be implemented in Project 2)
- Various embeddings of threading APIs in languages (C/POSIX threads, Java, C#)

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);
}
```

Race Conditions

“Too Much Milk” Problem

Person A	Person B
Look in the fridge: Out of Milk	
Leave for Wawa	
Arrive at Wawa	Look in fridge: Out of Milk
Buy milk	Leave for Wawa
Arrive home	Arrive at Wawa
	Buy milk
	Arrive home

- Don’t buy too much milk
- Any person can be distracted at any time

A possible solution?

Thread A

```
If (noMilk) {
    if (noNote)   {
          leave note
          buy milk
          remove note
    }
}
```

Thread B

```
If (noMilk) {
    if (noNote)   {
          leave note
          buy milk
          remove note
    }
}
```

- Does this method work?

Another possible solution?

Thread A

```
leave noteA
if (noNoteB) {
    if (noMilk) {
        buy milk;
    }
}
Remove noteA
```

Thread B

```
leave noteB
if (noNoteA) {
    if (noMilk) {
        buy milk;
    }
}
Remove noteB
```

- Does this method work?