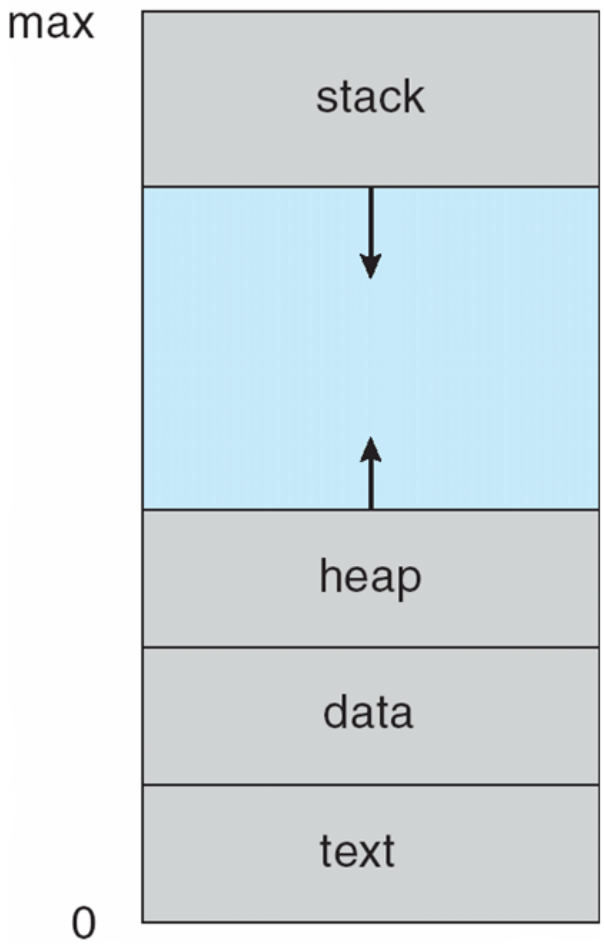
Processes and Threads

Process in Memory

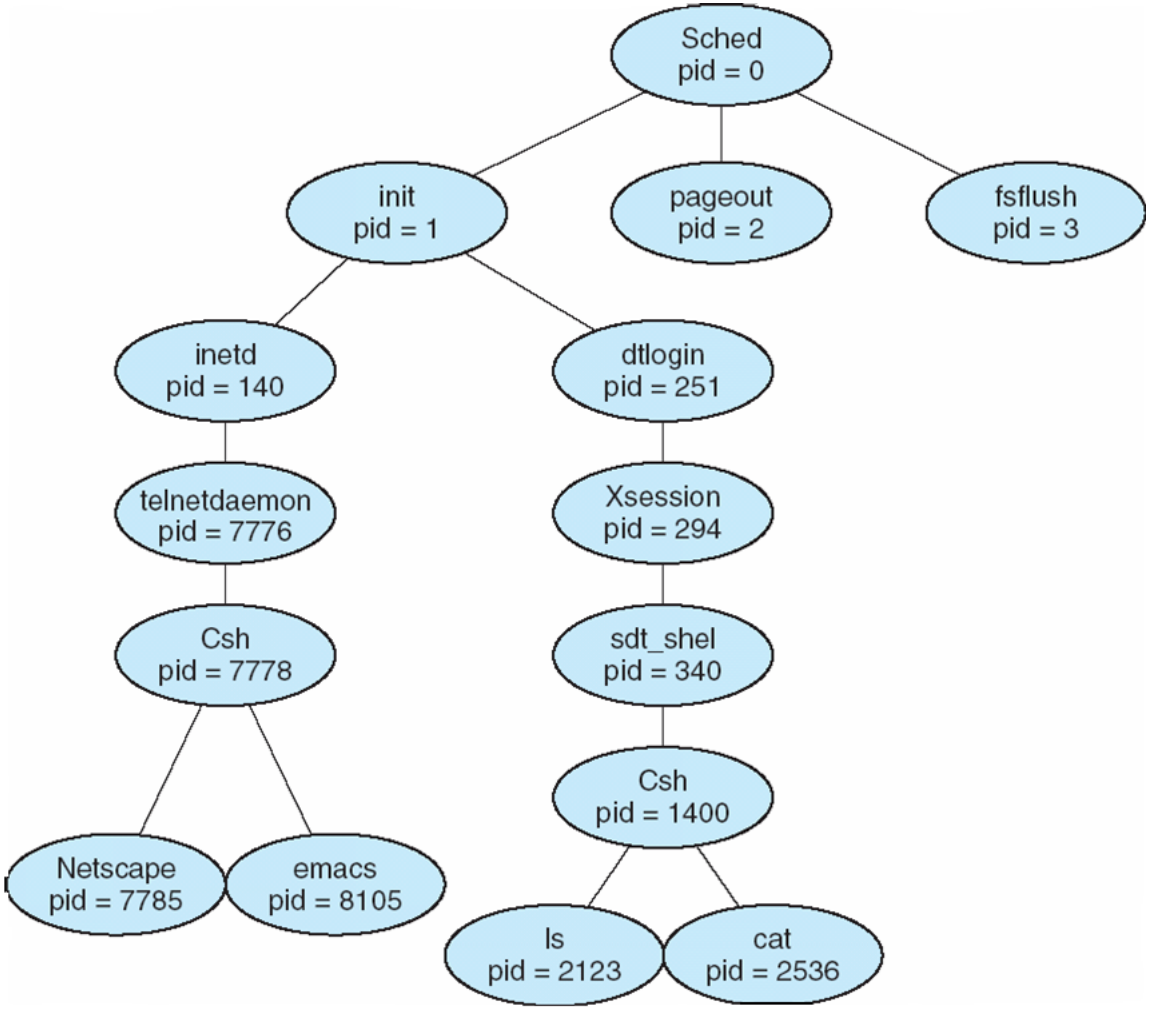
A process is a program in execution



Multiple processes

- Operating systems (OSes) run multiple processes on behalf of each user (example on next slide)
- Processes have a parent-child relationship
- Users can also write programs that use multiple processes – it is one kind of parallel programming
- Key function calls: fork(), family of exec() functions
- Fork and exec are system calls that each language provides a way to access
- It is easy to do in C/C++ since OSes are written in C

A tree of processes on a typical Solaris



C Program forking separate process

```
int main()
{
pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed");
          exit(-1);
    }
    else if (pid == 0) { /* child process */
          execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
          /* parent will wait for the child to
    complete */
          wait (NULL);
          printf ("Child Complete");
          exit(0);
    }
}
```

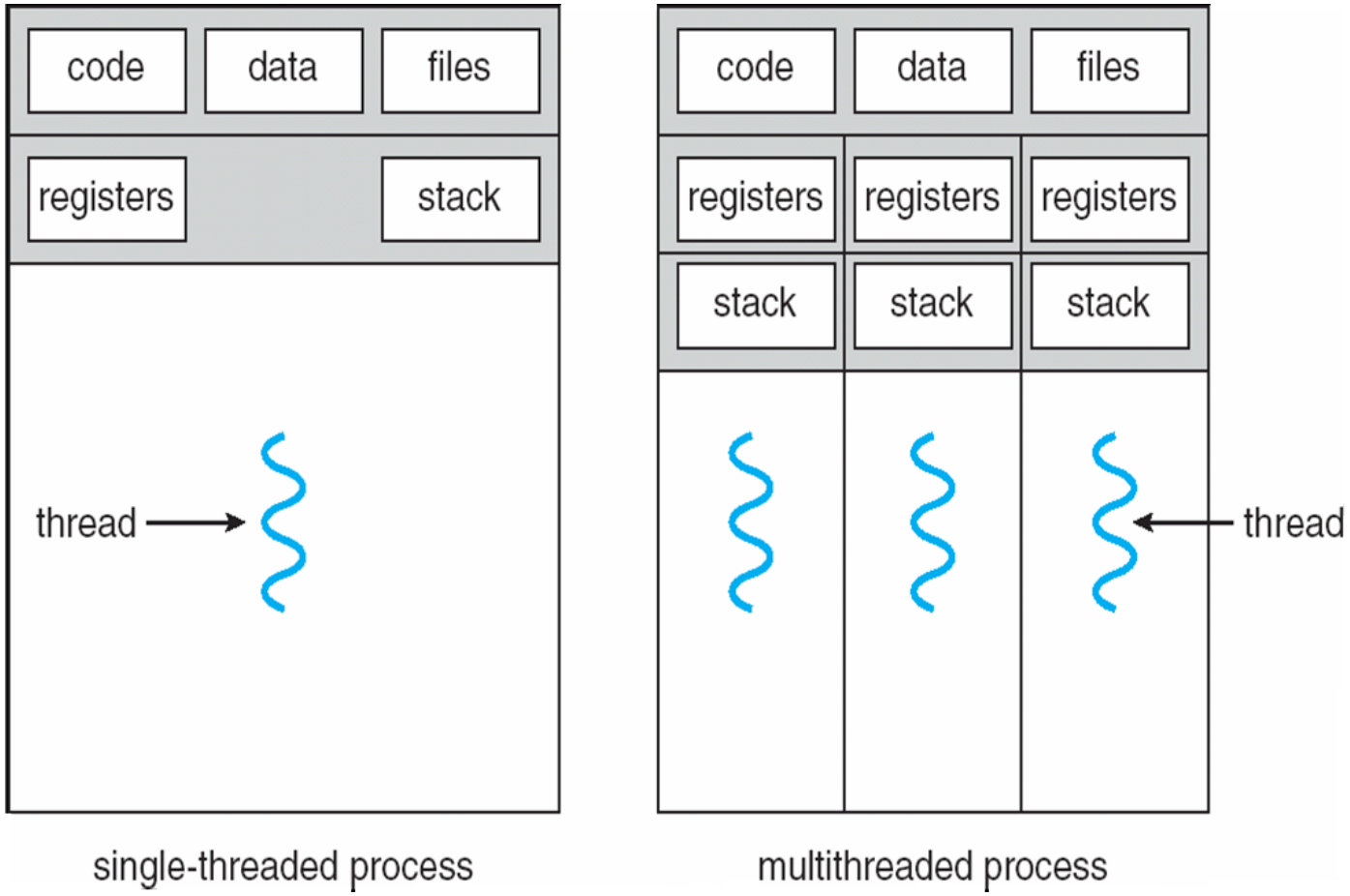
Why bother with parallel programming?

- Modern machines are multicore and parallel programming can utilize the idle cores
- If processing must stop due to network or disk I/O reasons, parallel programming allows other tasks to be completed in the meanwhile

Threads

- A fundamental unit of CPU utilization
- A thread is a baby process
- Yet another way of doing parallel programming
- Choice between using process vs thread is task specific
- Threads are lower overhead
- Threads share things (see next slide)

Single and Multithreaded Processes



single-threaded process

multithreaded process

Thread syncrhonization: Mutexes

- Mutex = mutual exclusion
- Mutexes are used for serializing shared resources such as memory
- Mutexes can be applied only to threads in a single process and do not work between processes
- Other primitives, such as, semarphores, exist for that purpose but outside the scope of this class

Threaded function without mutex

```
int counter = 0;
void functionC()
{
   counter++;
}
```

- Possible execution sequence for two threads:

Thread 1: counter = 0 → counter = 1;

Thread 2: counter = 0 → counter = 1;

- Undesirable, we need counter = 2;

Threaded function with mutex

```
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
void functionC() {
   pthread_mutex_lock(&mutex1);
   counter++;
   pthread_mutex_unlock(&mutex1);}
```

- Possible execution sequence for two threads:

Thread 1: counter = 0 → counter = 1;

Thread 2: counter = 0 → lockout → counter = 1;

- Threads 1 and 2 may be swapped in incrementing counter but counter = 2 when mutex is used

Pthread_mutex_trylock()

- What if multiple mutexes are in use and different threads end up locking different mutexes, potentially blocking each other?
- Such conditions are called deadlock conditions
- pthread_mutex_lock() blocks a thread until mutex is unlocked
- pthread_mutex_trylock() prevents deadlock by returning an error code to signal failure to acquire a lock

Thread pitfalls: race conditions

- Threads are scheduled by the operating system and are executed at random
- It cannot be assumed that threads are executed in the order they are created
- Threads may also execute at different speeds
- Mutexes and joins must be utilized to achieve a predictable execution order and outcome

Thread pitfalls: thread safe code

- The threaded routines must call functions which are "thread safe”
- This means that there are no static or global variables which other threads may clobber or read assuming single threaded operation
- If static or global variables are used then mutexes must be applied or the functions must be re-written to avoid the use of these variables

Thread pitfalls: mutex deadlock

- This condition occurs when a mutex is applied but then not "unlocked”
- This causes program execution to halt indefinitely
- It can also be caused by poor application of mutexes or joins
- Be careful when applying two or more mutexes to a section of code