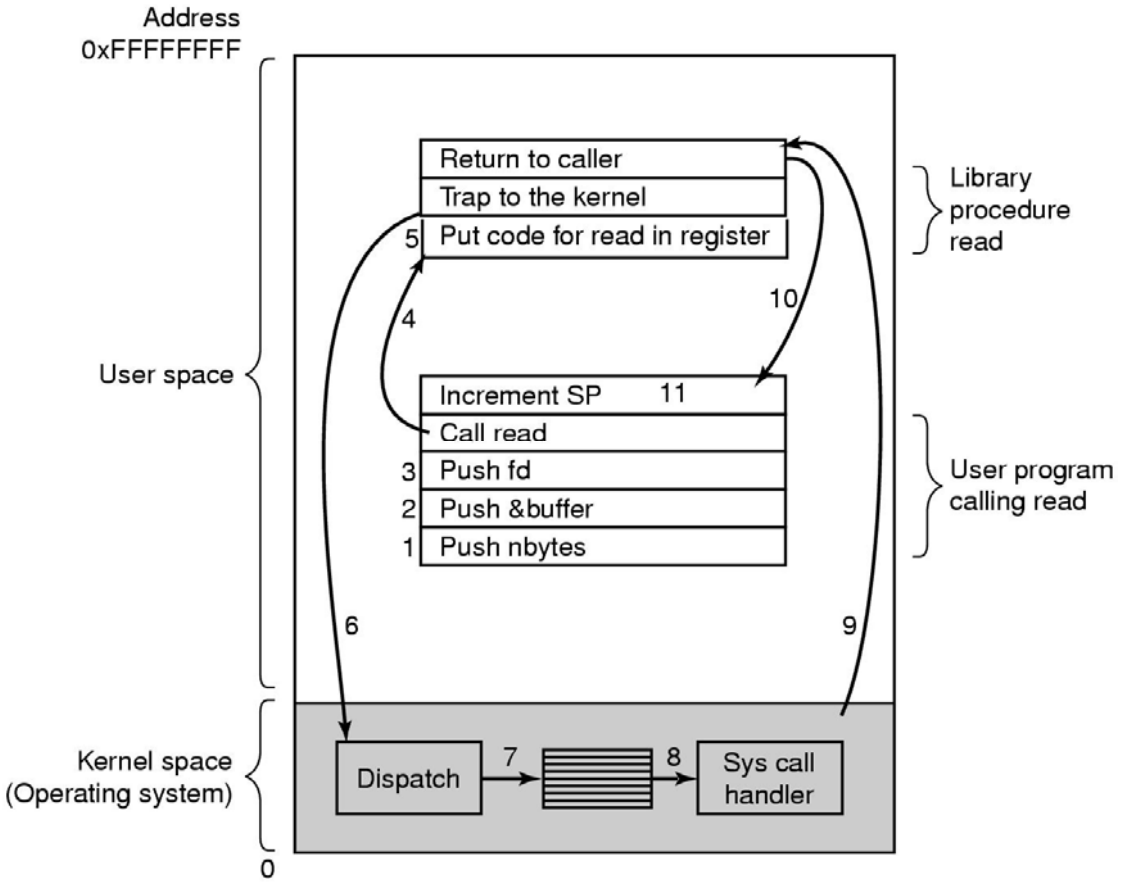
Realizing Concurrency using Posix Threads

B. Ramamurthy

System Call



There are 11 steps in making the system call

read (fd, buffer, nbytes)

Some System Calls For Process Management and File Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Pipe IPC (Answer to Jason's Ques)

- Pipe allows for arbitrary sequences of commands to be coupled together.
- Syscall pipe returns two file descriptors: one for writing fd[1] to the pipe and another for reading fd[0] from the pipe.

```
int fd[2], retval;
retval = pipe(fd);
```

On to threads..

Introduction

- A **thread** refers to a thread of control flow: an independent sequence of execution of program code.
- Threads are powerful. As with most powerful tools, if they are not used appropriately thread programming may be inefficient.
- Thread programming has become viable solution for many problems with the advent of multiprocessors and client-server model of computing.
- Typically these problems are expected to handle many requests simultaneously. Example: multi-media, database applications, web applications.

Topics to be Covered

- Objective
- POSIX threads
- What are Threads?
- Creating threads
- Using threads
- Summary

Objective

- To study POSIX standard for threads called Pthreads.
- To study thread control primitives for creation, termination, join, synchronization, concurrency, and scheduling.
- To learn to design multi-threaded applications.

Threads

- A thread is a unit of work to a CPU. It is strand of control flow.
- A traditional UNIX process has a single thread that has sole possession of the process's memory and resources.
- Threads within a process are scheduled and execute independently.
- Many threads may share the same address space.
- Each thread has its own private attributes: stack, program counter and register context.

Creating threads

- Always include pthread library:

#include <pthread.h>

- int pthread_create (pthread_t *tp, const pthread_attr_t * attr, void *(* start_routine)(void *), void *arg);
- This creates a new thread of control that calls the function start_routine.
- It returns a zero if the creation is successful, and thread id in tp (first parameter).
- attr is to modify the attributes of the new thread. If it is NULL default attributes are used.
- The arg is passing arguments to the thread function.

Using threads

1. Declare a variable of type pthread_t
2. Define a function to be executed by the thread.
3. Create the thread using pthread_create

Make sure creation is successful by checking the return value.

4. Pass any arguments need through' arg (packing and unpacking arg list necessary.)
5. #include <pthread.h> at the top of your header.
6. Compile:

g++ -o executable file.cc -lthread

Thread's local data

- Variables declared within a thread (function) are called local data.
- Local (static) data associated with a thread are allocated on the stack. So these may be deallocated when a thread returns.
- So don't plan on using locally declared variables for returning arguments. Plan to pass the arguments thru argument list passed from the caller or initiator of the thread.

Thread termination (destruction)

Implicit : Simply returning from the function executed by the thread terminates the thread. In this case thread's completion status is set to the return value.

- Explicit : Use thread_exit.

Prototype: void thread_exit(void *status);

The single pointer value in status is available to the threads waiting for this thread.

Waiting for thread exit

- int pthread_join (pthread_t tid, void **statusp);
- A call to this function makes a thread wait for another thread whose thread id is specified by tid in the above prototype.
- When the thread specified by tid exits its completion status is stored and returned in statusp.