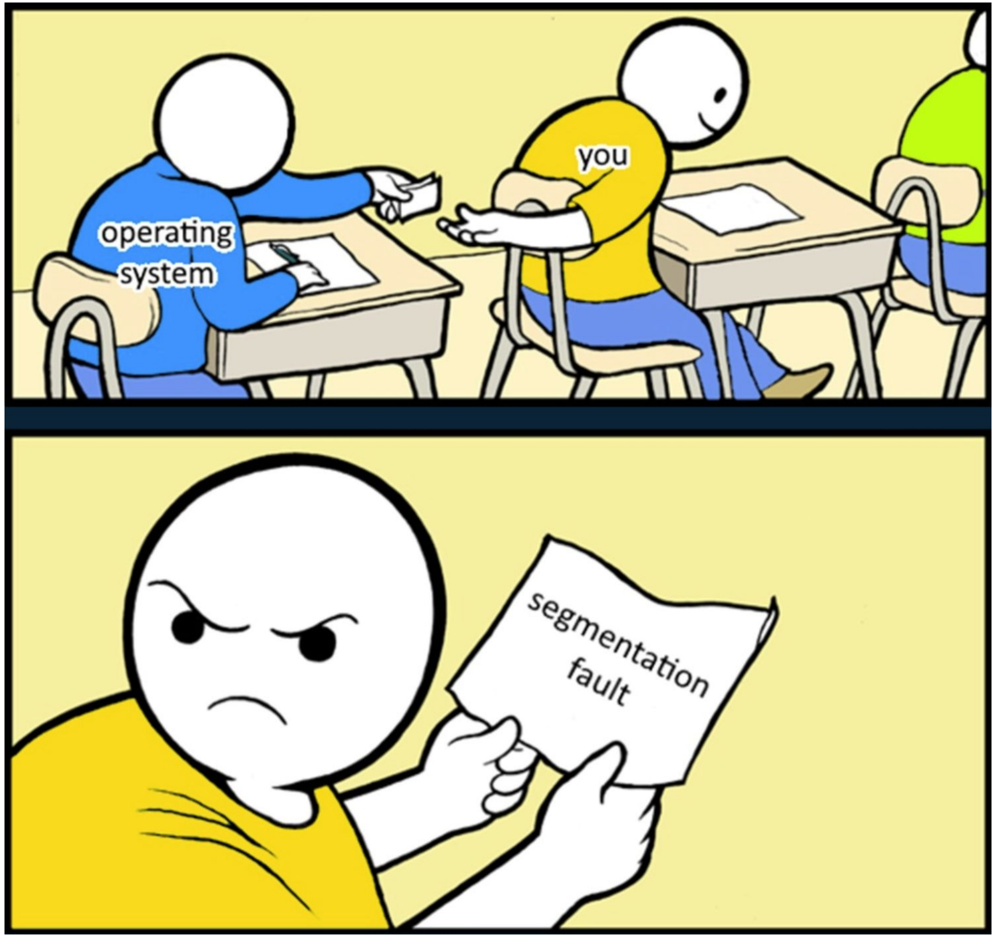
Meme of the day



CSCI 3752: Design & Analysis of Operating Systems

Recitation 6

Sreeram Ganesan

Today's Agenda

1. Inter Process Communication
2. Introduction to Pthreads
3. Work on PA2
4. No recitation assignment today!

I HAVE AN ANNOUCEMENT

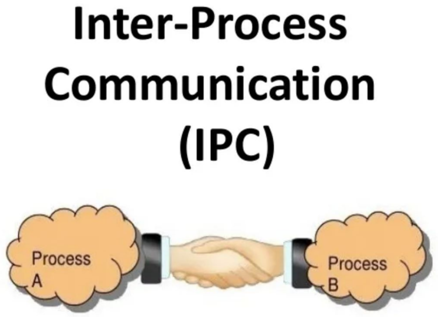
Announcements

1. Start working on PA2. Due Mar 5 at 11:59pm.
2. Recitation materials - shorturl.at/nxB24

1. Inter Process Communication (IPC)

1. IPC

1. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
2. The communication between these processes can be seen as a method of co-operation between them.
3. Processes can communicate with each other through:
   a. Shared Memory
   b. Message passing



1.1 Shared Memory

1. Shared Memory is an **efficient** means of passing data between programs. One process creates a shared memory segment that other processes (if permitted) can access. This is done by using `shmget` function.
2. In shared memory, multiple processes can access the same region of memory. This region of memory is typically created by one process, and other processes can attach to it. Once attached, these processes can read and write to the same memory location.
3. This method is faster than message passing because processes can directly access the data without any intermediate steps.
4. However, it also requires synchronization mechanisms to ensure that multiple processes do not access the same memory location at the same time, leading to race conditions.

`int shmget(key_t key, size_t size, int shmflg);`

PARAMETERS

- **key:** A key value used to identify the shared memory segment.
- **size:** The size, in bytes, of the shared memory segment.
- **shmflg:** A set of flags that determine the permissions and behavior of the shared memory segment. For example, the IPC_CREAT flag is used to create a new shared memory segment if it does not already exist. SHM_RDONLY is used to specifies that the shared memory segment should be read-only.

RETURN

- **Success:** A key value used to identify the shared memory segment
- **Error:** -1

1.1 Shared Memory

1. After the shared memory is created, it is attached to the process with `shmat`. Once attached, a process can read or write to the segment, as allowed by the permission requested in the attach operation.

   ```
   void *shmat(int shmid, const void *shmaddr, int shmflg);
   ```

2. `shmdt` detaches the shared memory segment located at the address indicated by `shmaddr`.

   ```
   int shmdt(const void *shmaddr);
   ```

1.2 Message Passing

1. In message passing, processes communicate with each other by sending and receiving messages via a communication channel. The communication channel can be a ***shared file***, a ***socket***, or a ***pipe***.
2. The sender process sends a message to the receiver process, which can then read and process the message. This method is slower than shared memory because it involves the overhead of copying data and context switching between processes.
3. However, it allows for better isolation between processes, as each process has its own memory space and cannot access the memory of other processes without explicit permission.

1.2.1 Linux Pipes

1. A pipe is a direct (in memory) I/O channel between two processes. Often it is used together with the system calls fork and exec to make multiple processes cooperate and perform parts of a larger task.
2. pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication.

`int pipe(int filedes[2]);`

PARAMETERS

- **`filedes`**: File descriptor array. The first file descriptor filedes[0] is used for reading from the pipe, and the second file descriptor filedes[1] is used for writing to the pipe.

RETURNS

- **Success:** 0
- **Error:** -1

1.3 Applications of Shared Memory & Message Passing

1. Shared memory is often used in high-performance computing applications, such as simulations and scientific calculations, where multiple processes need to access large amounts of data quickly. By using shared memory, processes can avoid the overhead of copying data between them.
2. Message passing is commonly used in distributed systems, such as client-server applications and network protocols. In these systems, processes communicate over a network by sending messages to each other. Message passing provides a way to ensure that messages are delivered reliably and in the correct order.

QUESTIONS

Code Walkthrough

Summary

1. Interprocess communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
2. Methods to perform IPC
   a. Shared Memory
   b. Message Passing
3. Shared memory is faster but requires synchronization, while message passing is slower but provides better isolation between processes. The choice between these two methods depends on the specific requirements of the application.

2. Pthreads

2.1 Thread

1. A thread is a basic unit of CPU utilization.
2. All the threads in a process share the same memory space and can access the same data structures, making them useful for implementing parallel algorithms and for improving the responsiveness of interactive applications.

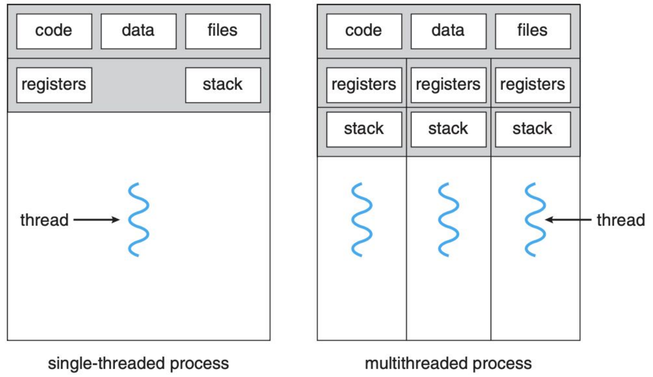


Figure 4.1 Single-threaded and multithreaded processes.

2.1 Thread

Each thread has its own unique

- Thread id
- Stack
- Set of registers
- Program counter

Advantages of having multiple threads in a programs

- **Improved performance:** Take advantage of multiple CPU cores
- **Improved responsiveness:** Don't block the main thread
- **Improved scalability:** Handle larger workloads or more users
- **Improved modularity:** Break your code into modules
- **Improved fault tolerance:** Recover from errors or failures more gracefully

2.2 What’s the catch with having multiple threads?

- **Race conditions:** When multiple threads access the same data concurrently, race conditions can occur. A race condition happens when the behavior of a program depends on the order in which threads execute. This can lead to unpredictable and hard-to-debug behavior.
- **Deadlocks:** A deadlock occurs when two or more threads are waiting for each other to release a resource, such as a lock or a semaphore. This can cause the threads to become stuck in an infinite loop, unable to make progress.
- **Starvation:** Starvation occurs when a thread is prevented from making progress because it cannot acquire a required resource. This can happen when other threads are hogging the resource, or when the resource is being used inefficiently.

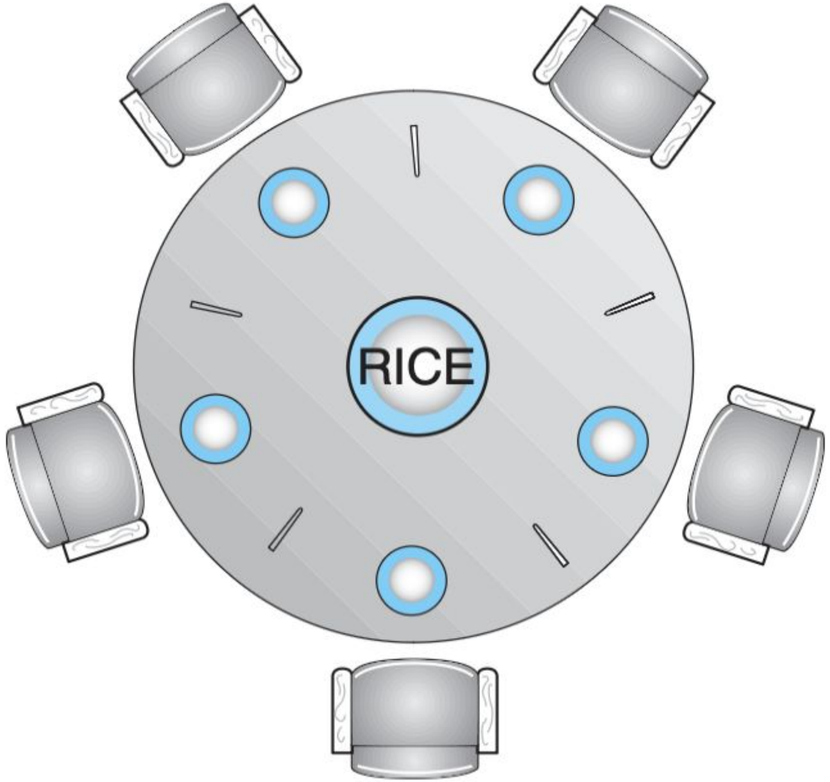


Figure 5.13 The situation of the dining philosophers.

2.3 Pthread

1. The POSIX thread libraries are a standards based thread API for C/C++.
2. Pthreads provides a set of C language programming interfaces for creating and manipulating threads, as well as for managing thread synchronization and communication.
3. It is most effective on multi-processor or multi- core systems where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing.
4. While most effective on a multiprocessor system, gains are also found on uniprocessor systems which exploit latency in I/O and other system functions which may halt process execution. (One thread may execute while another is waiting for I/O or some other system latency.)
5. Pthreads can also be used for implementing parallel algorithms that can perform computations on large data sets in parallel.

2.3.1 Pthread Creation

1. A thread is spawned by defining a function and its arguments which will be processed in the thread.
2. `pthread_create( )` is the function that creates a thread.
3. `#include <pthread.h>` for using Pthread APIs.

`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`

PARAMETERS

- `thread:` A pointer to a `pthread_t` variable that will be used to identify the new thread
- `attr:` A pointer to a `pthread_attr_t` structure that specifies the attributes of the new thread. This can be set to `NULL` to use default attributes.
- `start_routine:` A pointer to the function that the new thread will execute.
- `arg:` A pointer to an argument that will be passed to the `start_routine` function.

RETURNS

- Success: 0
- Error: Error code is returned on failure. The error codes are defined in the `errno.h` header file, and can include `EAGAIN` (insufficient resources to create another thread), `EINVAL` (invalid thread attributes), and `EPERM` (insufficient privileges to set the scheduling policy and parameters).

2.3.2 Pthread Attributes

1. pthread_attr_t is a structure that contains the attributes of a thread. These attributes include the stack size, scheduling policy, and priority, among others. The pthread_attr_t structure is used by the pthread_create() function to specify the attributes of the new thread being created.
2. The pthread_attr_t structure can be initialized using the pthread_attr_init() function. This function initializes the structure with default values. The attributes can then be modified using other functions, such as pthread_attr_setstacksize() and pthread_attr_setschedpolicy().
3. Types:
   a. Thread Creation Attributes
   b. Thread Scheduling Attributes

2.3.2 Pthread Attributes

Creation

```
int pthread_attr_init(pthread_attr_t *attr);

int pthread_attr_destroy(pthread_attr_t *attr);

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t
sz);

int pthread_attr_getstacksize(const pthread_attr_t *attr,
size_t *sz);

int pthread_attr_setstack(pthread_attr_t *attr, void *addr,
size_t stacksize); int pthread_attr_getstack(const
pthread_attr_t *attr, void **addr, size_t  *stacksize);
```

Scheduling

```
int pthread_attr_setscope(pthread_attr_t
*attr, int scope);

int pthread_attr_getscope(const
pthread_attr_t *attr, int *scope);
```

QUESTIONS

Code Walkthrough

Summary

1. A thread is a basic unit of CPU utilization. All threads in a process share the same address space.
2. Each thread has its own unique - Thread id, Stack, Set of registers, Program counter
3. Adv - performance, responsiveness, scalability, modularity, fault tolerance
4. DisAdv - Race Condition, Deadlock, Starvation
5. Pthreads(POSIX Threads) are standard based thread API in C/C++ that provides a set of programming interfaces for creating and manipulating threads, as well as for managing thread synchronization and communication.