CSCI 4061 Lab #13

April 25, 2022

Agenda

Exam #2 review

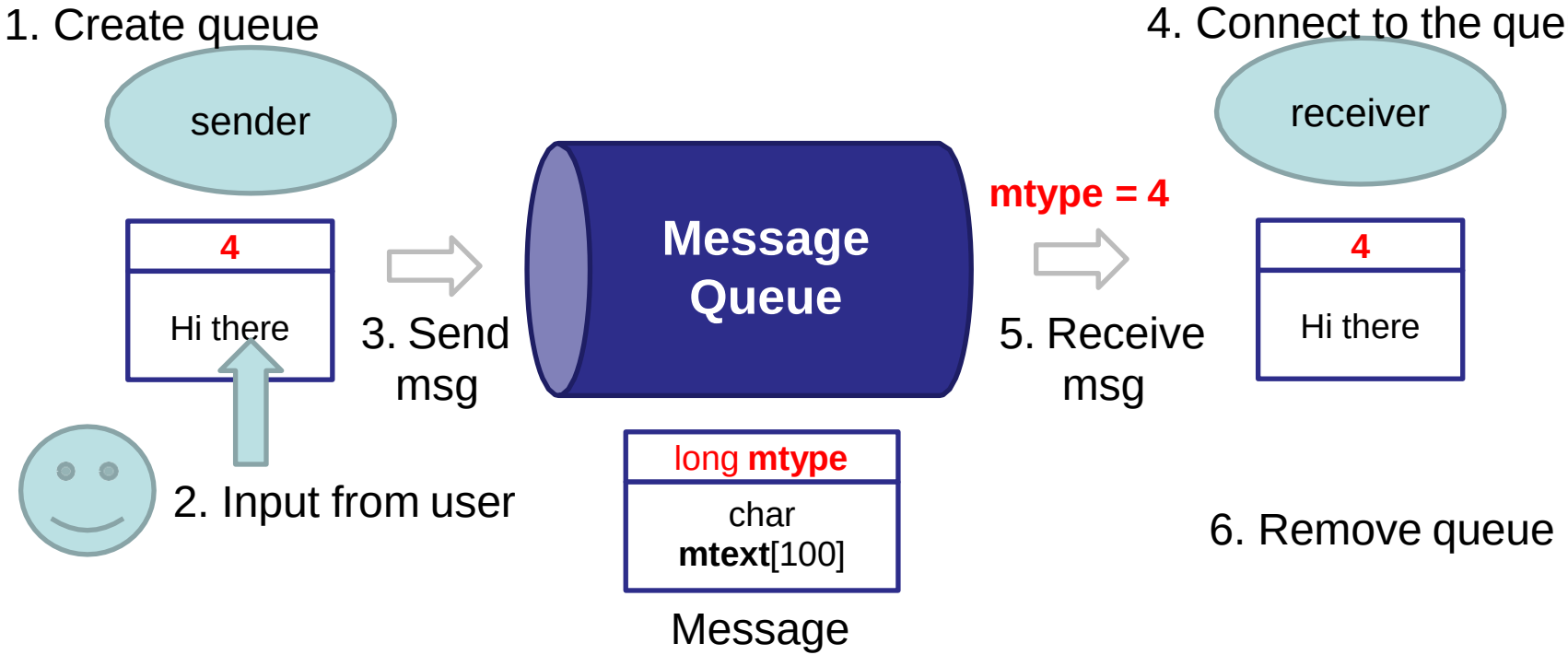
- IPC: Message Passing and Shared Memory
- Signals
- Threads
- Synchronization
- Networking/Socket Programming

No Exercise

Message Passing

- Sender puts messages into queue.
- Receiver pulls the messages out of the queue.
- POSIX API
  - **msgget()**: Create a message queue or connect to an already existing message queue
  - **msgsnd()**: Write into message queue
  - **msgrcv()**: Read from the message queue
  - **msgctl()**: Perform control operations on the message queue

Message Queue example



Shared Memory

- Shared-memory allows two or more processes to share a segment of physical memory.
- POSIX API
  - **shmget():** create shared memory segment
  - **shmat()**: attach to the shared memory segment
  - **shmdt()**: detach from the shared memory segment
  - **shmctl()**: remove shared memory segment

Signals

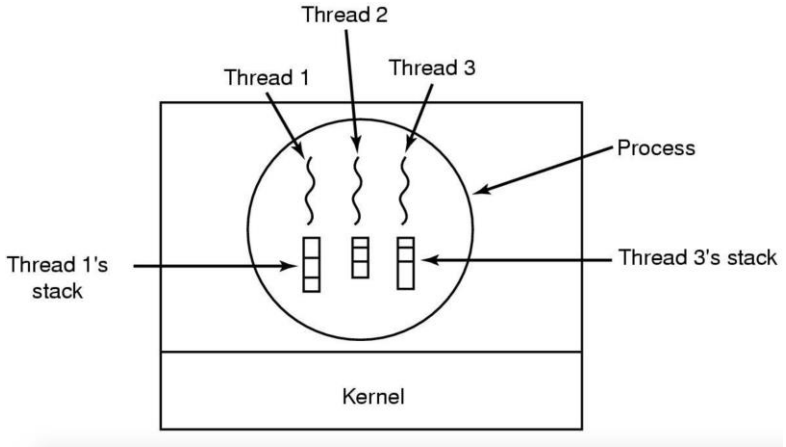
- Signals are a form of *asynchronous* IPC
  - Software interrupt to a process
- Examples:
  - **SIGABRT:** Process abort
  - **SIGINT:** Ctrl-C
  - **SIGSTOP:** Execution stopped
  - **SIGKILL/ SIGTERM:** Terminated
  - **SIGCHLD:** Child exited
- Send signals using **kill()**

Handling Signals

- Default action (most cases will cause termination)
- Ignore (protect against ^C)
- Block signals : queued in OS
  - The process receives the signal only after it is unblocked
  - **sigprocmask()**
- Take specific action/handle
  - Associate an action on delivery of signal
  - **sigaction()**

Threads

- An abstraction for an executing instruction stream
- Multiple threads share resources within a process.
  - Have their own stacks
- Joinable threads and detachable threads



Threads: Benefits and Drawbacks

- Benefits:
  - **Concurrency:** multiple pieces of code can be executed simultaneously
  - **Modularity:** decompose functionality
  - **Parallelism:** threads running in parallel (multi-core)
  - **Scale:** more threads available than processors
  - **Overhead:** multiple threads cheaper than multiple processes
- Drawbacks:
  - **Sharing and thread safety:** requires synchronization
  - **Global variables:** per thread global variables may be required

Threads POSIX API

- **pthread_create():** create a thread
- **pthread_self():** identify the calling thread
- **pthread_equal():** compare two threads
- **pthread_exit():** exit the thread
- **pthread_cancel():** cancel a thread
- **pthread_join():** wait for a joinable thread to complete

Synchronization: Locks

- Mutex is a lock that we set before using a shared resource and release after using it.
  - This helps in synchronization
- POSIX API:
  - pthread_mutex_init(): initializes the lock
  - pthread_mutex_lock(): acquire lock if available else wait until available
  - pthread_mutex_unlock(): release the lock

Condition Variables

- Method of blocking while waiting for a condition to be satisfied
- A thread blocks itself when the condition is false
  - A different thread wakes it when the condition is true
- POSIX API:
  - **pthread_cond_init():** initialize a condition variable
  - **pthread_cond_wait():** wait on a condition variable
  - **pthread_cond_signal():** signal one thread waiting on that condition variable
  - **pthread_cond_broadcast():** signal all threads waiting on that condition variable

Semaphores

- Does not require busy waiting.
- POSIX API:
  - **sem_init():** initialize a semaphore
  - **sem_wait():** wait on a semaphore / decrement
  - **sem_post():** post to a semaphore / increment

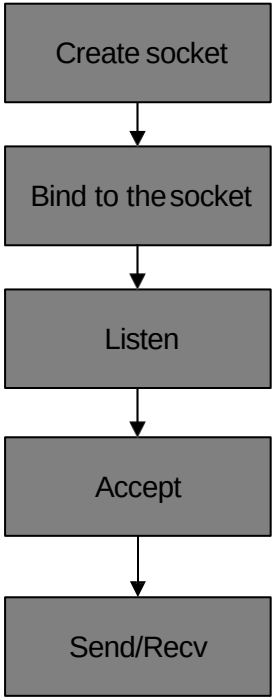
Networking

- Enable communication between processes on different machines.
- Socket –abstraction of a communication endpoint
  - Process sends/receives messages to/from its socket
- Transport protocols
  - TCP: Connection-oriented, reliable
  - UDP: Connectionless, unreliable

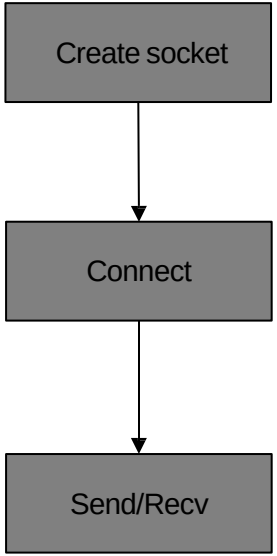
Client-Server Architecture

- Server process waits to be contacted
- Client process initiates communication

Server side



Client Side



POSIX API

- **int socket(int** *domain*, **int** *type*, **int** *protocol*)**:** Creates a socket
- **int bind(int** *sockfd*, **const struct sockaddr ***addr* **socklen_t** *addrlen*)
  - Bind the socket to a network address
- **int listen(int** *sockfd*, **int** *backlog*)**:** Listen for incoming connection requests
- **int accept(int** *sockfd*, **struct sockaddr ***addr*, **socklen_t ***addrlen*)
  - Accept a new connection, blocks until a new request arrives.
- **int connect(int** *sockfd*, **const struct sockaddr ***addr* **socklen_t** *addrlen*)
  - Connect to the server
- **ssize_t sendto(int** *sockfd*, **const void ***buf*, **size_t** *len*, **int** *flags* **const struct sockaddr ***dest_addr*, **socklen_t** *addrlen*)**:** Send data
- **ssize_t recvfrom(int** *sockfd*, **void ***buf*, **size_t** *len*, **int** *flags* **struct sockaddr ***src_addr*, **socklen_t ***addrlen*)**:** Receive data

Questions?

Thank you! All the best!