Unix Inter-process Communication

Chris Kauffman

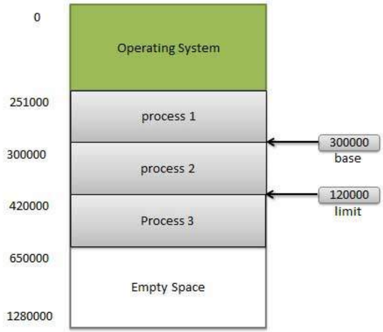
CS 499: Spring 2016 GMU

Mini-exam 2 back

Stat	Val	
Mini-exam 2		
Count	32	
Average	35.84	89.6%
Median	36.00	90.0%
Standard Deviation	3.45	8.6%
Mini-exam 1		
Count	34	
Average	35.44	88.6%
Median	36.50	91.3%
Standard Deviation	3.32	8.3%

Results overall good (again)

Basic Process Architecture



Source: Tutorials Point

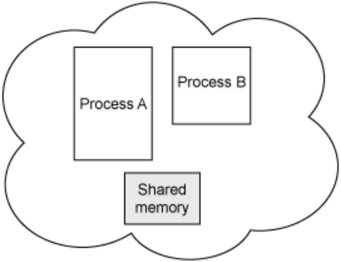
- Separate Memory Image for Each Process
- OS + Hardware keeps processes inside their own address space

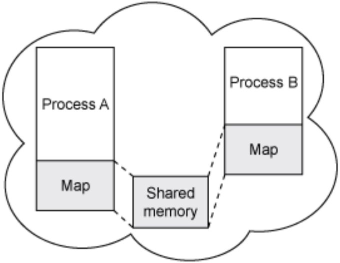
Unix Interprocess Communication

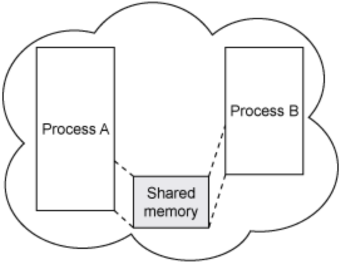
- Single Machine
- Controlled mechanisms for one process to pass info to another
- Simple: Pipes
- Moderate: Message Queues
- Complex: Shared memory with semaphores (locks)
- Complex involves IPC library calls, centralized authority (OS) to manage shared resources like queues, shmem

Use of Shared Memory Resources

1. Single proc creates shard memory area
2. Multiple procs attach/map local address to shared memory
3. IPC via shared memory now possible







Two Distinct Flavors of Unix IPC

System V

- Older, somewhat more archaic
- Widely implemented, many existing codes based on it
- May not be thread safe

POSIX

- Newer, simpler interfaces
- Not as widely implemented
- Thread Safe

Both Provide Similar Basic Tools

- Message Queues: Basic send/receive
- Semaphores: Atomic get/set with blocking
- Shared Memory: Raw arrays of shared data
- Additional differences on StackOverflow

Focus for the Moment on System V

- Will visit POSIX stuff via POSIX threads
- Just want rough overview anyway

Semaphores

- General purpose locking mechanism
- Atomic operations to decrement/increment
- Typically allocate an array of semaphores
- IPC allows atomic operation on multiple semaphores in the array simultaneously: useful for dining philosophers

Activity: Revisiting the Philosophers

Examine the dining philosophers code here:
`https://cs.gmu.edu/~kauffman/cs499/philosophers.c`
Use the IPC guide here: `http://beej.us/guide/bgipc/output/html/singlepage/bgipc.html`
Find out how the following are done:

- Spawn a new process
- Determine child/parent
- What is a semaphore?
- How does one get a semaphore?
- What does one do with a semaphore?

Lessons Learned from `philosophers.c`

- `fork()` is used to create new processes, clones of the parent save for the return value of `fork()` call which is child PID in the parent and 0 for the child
- `int semid = semget(...);` is used to obtain a semaphore from the operating system which returns an integer id of a semaphore. Options allow retrieval of an existing semaphore or creation of a new one.
- System V semaphores are arrays of counters and operations must specify which element in the array is operated upon
- On creation, the values in the semaphore are undefined and must be specified.
- `semctl()` is used to get and set values from the semaphore which is done atomically but cannot be used to increment/decrement values
- `semop()` is used to atomically increment/decrement values in the semaphore and requires use of a `struct sembuf`
- Processes can attempting to decrement a semaphore below 0 will block and wait until its value returns becomes positive.

The Nature of a Semaphore

SO: cucufrog on Condition Variables vs Semaphores

A condition variable is essentially a wait-queue, that supports blocking-wait and wakeup operations, i.e. you can put a thread into the wait-queue and set its state to BLOCK, and get a thread out from it and set its state to READY.

- Requires use of a mutex/lock in conjuction

A Semaphore is essentially a counter + a mutex + a wait queue.

- It can be used as it is without external dependencies.
- You can use it either as a mutex or as a conditional variable.

Message Queues

- Implements basic send/receive functionality through shared memory
- Similar to MPI: one process sends, another receives
- Atomic access/removal taken care of for you
- Allow message filtering to take place based on a tag

Kirk and Spock: Talking Across Interprocess Space

- Demo the following pair of simple communication codes which use System V IPC Message Queues.
- Examine source code to figure out how they work.

https://cs.gmu.edu/~kauffman/cs499/kirk.c
https://cs.gmu.edu/~kauffman/cs499/spock.c

Viewing Shared System Resources

Shared memory resources can outlast the program which created them. The following unix commands are useful for manipulating them from the command line.

```
ipcs (1)  - show information on IPC facilities
ipcrm (1) - remove certain IPC resources
ipcmk (1) - make various IPC resources
```

Mostly `ipcs` to list, `ipcrm` to clean up when something has gone wrong.

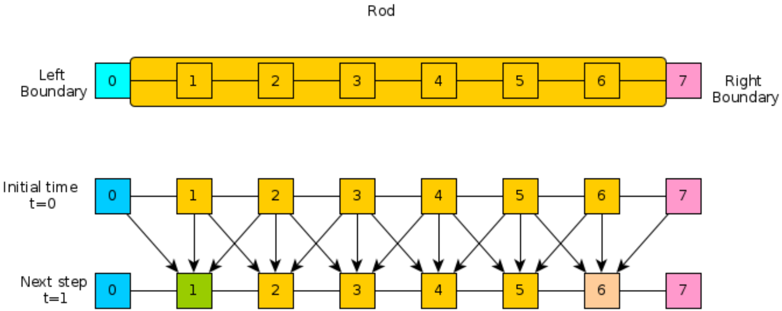
System V IPC Shared Memory Segments

- The ultimate in flexibility is to get a segment of raw bytes that can be shared between processes
- Examine `shmdemo.c` to see how this works
- Importantly, this program creates shared memory that outlives the program

`https://cs.gmu.edu/~kauffman/cs499/shmdemo.c`

Recall Heat

- Finite element simulation of a 1D rod, fixed heat reservoirs at both ends
- Calculate 2D Array of heat values over time, each row is a single time step



Share the Warmth: Sys V IPC for Heat

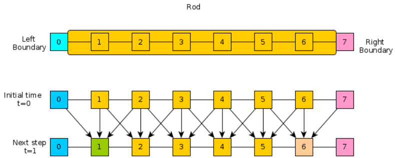
Construct a plan to use them to simulate the heated rod from earlier in the class.

```
// Make a new process
int pid = fork(..);

// Get+manipulate semaphores
int semid = semget(key,...);
semctl(semid, i, GETVAL);
semctl(semid, i, SETVAL, 1);
op.sem_op = -1;
op.sem_num = index;
semop(semid, &op, 1);

// get+manipulate message queues
int msqid = msgget(key, ...);
msgsnd(msqid, &buf, ...);
msgrcv(msqid, &buf, ...);

// get/attach shared memory
int shmid = shmget(key);
int *data = shmat(shmid,..);
```



Two IPC Heat Designs

Both

- Divide the Heat matrix into column blocks owned by each processor
- Each proc works on its own block
- Communicates with neighboring processors to calculate boundary elements

Like MPI Version

- Very little data shared between processes
- Use message queues to coordinate work

Like a Shared Memory Version

- Use a hunk of shared memory
- Use semaphores or message queues to coordinate multiple processes

More Resources

http://www.tldp.org/LDP/tlk/ipc/ipc.html