CSCI 4061: Inter-Process Communication

Chris Kauffman

Last Updated:
Tue Nov 7 12:34:27 CST 2017

Logistics

Reading

- Stevens/Rago
  Ch 15.6-12
- Robbins and Robbins
  Ch 15.1-4

Goals

- Protocols for Cooperation
- Basics of IPC
- Semaphores, Message Queues, Shared mem

Lab08: FIFO, protocol

How did it go?

Project 2

- Kauffman not happy with delay
- You will be happier with result

Exercise: Forms of IPC we've seen

- Identify as many forms of inter-process communication that we have studied as you can
- For each, identify restrictions
  - Must processes be related?
  - What must processes know about each other to communicate?
- You should be able to name at least 3-4 such mechanisms

Answers: Forms of IPC we've seen

- Pipes
- FIFOs
- Signals
- Files

Inter-Process Communication Libraries (IPC)

- FIFOs allow info transfer between unrelated processes
- Common patterns exist in IPC, met with IPC libraries which include
  1. Semaphores: counters with locking and wait queues
  2. Message queues: direct-ish communication between processes
  3. Shared memory: array of bytes accessible to multiple processes
- Two flavors of these IPC
  1. System V IPC: older, widely implemented, dated
  2. POSIX IPC: newer, mostly implemented, improved

Additional differences on StackOverflow

Which Flavor of IPC?

System V IPC (XSI IPC)

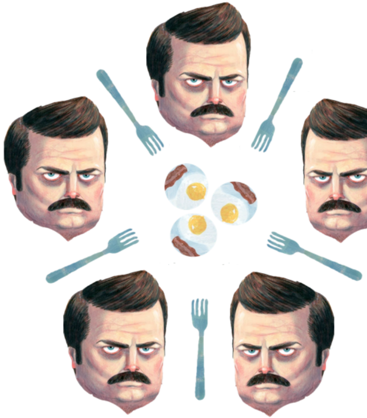
- Most of systems have System V IPC but it's kind of strange, has its own *namespace* to identify shared things
- Part of Unix standards, referred to as XSI IPC and may be listed as optional
- Most textbooks/online sources discuss some System V IPC. Example:
  - Stevens/Rago 15.8 (semaphores)
  - Robbins/Robbins 15.2 (semaphore sets)
  - Beej's Guide to IPC

POSIX IPC

- POSIX IPC little more regular, uses filesystem to identify IPC objects
- Originated as optional POSIX/SUS extension, now required for compliant Unix
- Covered in our textbooks partially. Example:
  - Stevens/Rago 15.10 POSIX Semaphores
  - Robbins/Robbins 14.3-5 POSIX Semaphores

Model Problem: Dining "Philosophers"

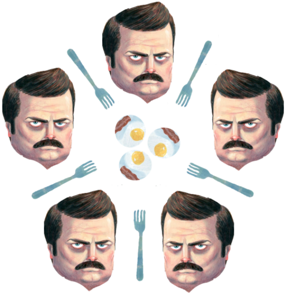
- Each Swansons will only eat with two forks
- JJ's only has 5 forks, must share
- After acquiring 2 forks, a Swanson eats an egg, then puts both forks back to consider how awesome he is
- Algorithms that don't share forks will lead to injury



Source: Aditya Y. Bhargava,

Exercise: Protocol for Dining "Philosophers"

- Each Swansons will only eat with two forks
- JJ's only has 5 forks, must share
- Swanson's pick up one fork at a time from left or right
- After acquiring 2 forks, a Swanson eats an egg
- After eating an egg a Swanson puts both forks considers how awesome he is, repeats
- After eating sufficient eggs, Swanson leaves
- Is there any potential for deadlock? How can this be avoided?
- Is there any chance for starvation?



Answer: Protocol for Dining "Philosophers"

All get Left Fork first: Deadlock

- Each Swanson can acquire 1 fork
- Waits forever for right fork

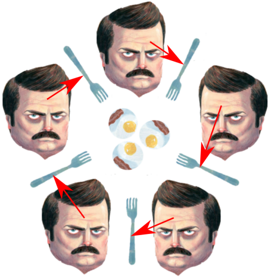
One goes Right first: Viable

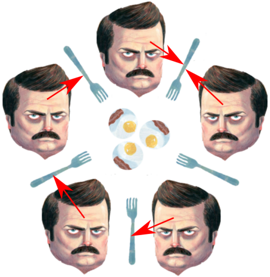
- Breaks the cycle so deadlock is not possible - A *viable* solution

Starvation?

- Give up both forks after eating an egg, others can get them, everyone eats *eventually*
- Some may wait until others completely finished: bad. Improve by giving up one fork if can't get the other

Deadlock





Viable

Semaphore

Source: Wikipedia Railway Sempahore Signal

- A counter variable with atomic operations
- Atomic operation: not divisible, all or none, no partial completion possible
- Used to coordinate access to shared resources such as shared memory, files, connections
- Typically allocate an array of semaphores
- IPC allows atomic operation on multiple semaphores in the array simultaneously: useful for dining philosophers

Activity: Dining "Philosophers" with Semaphores

Examine the dining philosophers code here:
`http://www.cs.umn.edu/~kauffman/4061/philosophers.c`
Use the IPC guide here:
`http://beej.us/guide/bgipc/output/html/singlepage/bgipc.html`
Find out how the following are done:

1. What does a C semaphore look like?
2. How does one create a semaphore?
3. How does `semop()` work, its arguments and behavior?
4. Are there any restrictions on values a semaphore can hold?
5. What happens when multiple processes modify the same semaphore?
6. How are semaphores used to coordinate the start of the meal?
7. How can a semaphore be used to coordinate use of forks?

Lessons Learned from `philosophers.c`

- `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 [process or] thread into the wait-queue and set its state to BLOCK, and get a thread out from it and set its state to READY.

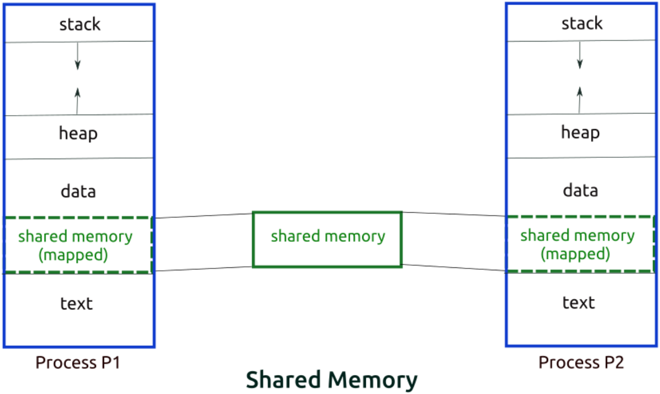
- Requires use of a mutex/lock in conjuction

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.

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: must clean it up at some point



Source: SoftPrayog System V IPC

Viewing Shared System V IPC 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.

Exercise: Email lookup with Shared Memory

- In lab, worked on a simple email lookup "server" or database
- Clients connected to server, server gave back emails based on name
- Shared memory makes server/client less relevant
- Propose how to use shared memory for email lookups AND alterations
- How might multiple processes coordinate use of shared memory?

```
// structure to store a lookup_t of
// name-to-email association
typedef struct {
  char name [STRSIZE];
  char email[STRSIZE];
} lookup_t;

lookup_t original_data[NRECS] = {
  {"Chris Kauffman"       ,"kauffman@umn.edu"},
  {"Christopher Jonathan" ,"jonat003@umn.edu"},
  {"Amy Larson"           ,"larson@cs.umn.edu"},
  {"Chris Dovolis"        ,"dovolis@cs.umn.edu"},
  {"Dan Knights"          ,"knights@cs.umn.edu"},
  {"George Karypis"       ,"karypis@cs.umn.edu"},
  ...
```

```
# Sample of potential use
> email_db lookup 'Chris Kauffman'
Looking up Chris Kauffman
Found: kauffman@umn.edu
> email_db lookup 'Rick Sanchez'
Looking up Rick Sanchez
Not found
> email_db change 'Chris Kauffman' 'kman@kauffmoney.co
Changing Chris Kauffman to kman@kauffmoney.com
Alteration complete
> email_db lookup 'Chris Kauffman'
Looking up Chris Kauffman
Found: kman@kauffmoney.com
```

Answer: Email lookup with Shared Memory

- Store entire array of name/email in a piece of shared memory with a know key
- Processes needing it attach to shared memory, scan through looking
- Updates can be done by altering the shared memory
- Danger multiple processes writing may corrupt the data
- Use semaphores to control access for reading/writing, would need to establish a protocol for this

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.

```
10-ipc-code/kirk.c
10-ipc-code/spock.c
```

Unique Identifiers in IPC: `ftok(char*,char)`

- System V IPC uses the notion of keys and IPC ids so unrelated processes can find shared resources
- Both `kirk.c` and `spock.c` use the same arguments to find the right message queue

```
key_t key = ftok("kirk.c", 'B');
int msqid = msgget(key, 0644 | IPC_CREAT);
```

- Key is tied to a specific known file which participating processes all know about
- Involves using new symbols like `IPC_CREAT` etc.

> *These IPC features were later added to System V. They are often criticized for inventing their own namespace instead of using the file system.*
> *– Stevens/Rago 15.6 XSI IPC*

- POSIX IPC create/open interface is closer to standard Unix I/O open/close operations

```
int flags = O_RDWR | O_CREAT;
int perms = S_IRUSR | S_IWUSR;
mqd_t msg_queue = mq_open("kirk.c", flags, perms);
```

Email Lookup with Message Queues

- Email lookup server from lab used FIFOs for server and clients to talk
- Would not be too hard to rewrite this with message queues
- Message queues allow filtering of messages, easy to direct at a specific process
- Get automatic blocking and resuming when receiving messages so don't need explicit signals
- Will be the subject of next Lab

More Resources on IPC

- `http://beej.us/guide/bgipc/`
- `http://www.tldp.org/LDP/tlk/ipc/ipc.html`