

Unix IPC Mechanisms

Alexander Zangerl az@bond.edu.au

Bond University

pipe

- ▶ is a one-way synchronous communication channel
- ▶ works between related processes (ie. parent-child)
- ▶ some data is buffered in pipe
- ▶ reading from empty pipe blocks, also writing to full pipe
- ▶ pipes are setup in pairs (one write, one read), by parent!
- ▶ prototype:
 - ▶ `int pipe(int fd[2]);`
- ▶ `fd[0]` is for reading, `fd[1]` for writing
- ▶ fds are low-level, can use `fdopen()` to make file stream

named pipe, FIFO

- ▶ similar to pipe in function
- ▶ but accessed as part of the file system
- ▶ result:
 - ▶ independent processes can use it
- ▶ unidirectional, blocking operation
- ▶ opening blocks until other side also opens
- ▶ prototype:
 - ▶ `int mkfifo (const char *pathname, mode_t mode);`
- ▶ see also:
 - ▶ `man 4 fifo (linux)`, `man -s 3C mkfifo (solaris)`

Unix Domain Sockets

- ▶ communication local on a single system
- ▶ but using network access functions
- ▶ prototype:
 - ▶ `unix_socket = socket(PF_UNIX, type, 0);`
- ▶ type arg holds address information
- ▶ access with `send()`, `recv()`, `close()`
- ▶ see also:
 - ▶ `man 7 unix (linux)`

memory map

- ▶ maps a file into memory
- ▶ also works on shared memory
- ▶ prototype:
 - ▶ `void *mmap(void *start, size_t len, int prot, int flags, int fd, off_t offset);`
 - ▶ `int munmap(void *start, size_t len);`
- ▶ multiple processes can map same file concurrently
- ▶ writing to mmap location is same as writing to file
- ▶ no synchronisation or locking!
- ▶ flag `MAP_ANON`: makes mmap not use file

SystemV shared memory

- ▶ attach shared memory to process
- ▶ multiple processes can attach same region
- ▶ region identified via numeric key
- ▶ prototype:
 - ▶ `int shmget(key_t key, int size, int shmflg);`
 - ▶ `int shmctl(int shmid, int cmd, struct shmid_ds *buf);`
- ▶ access is unsynchronised!

dotfile locking

- ▶ advisory only
- ▶ for file X produce file .X
- ▶ simple, no OS support needed
- ▶ works because file creation is atomic (process's perspective)
- ▶ but lockfile is not removed when process exits
- ▶ convention: put PID in dotfile
- ▶ useful to clean stale locks

lockf locking

- ▶ prototype:
 - ▶ `int lockf(int fd, int cmd, off_t len);`
- ▶ produces lock on (part of) file
- ▶ further attempts: either block the caller or fail
- ▶ lock is removed on process exit
- ▶ `lockf()` is POSIX, BSD variant is `flock()`

semaphores

- ▶ protected, shared variable
- ▶ counter for resource access synchronisation
- ▶ operations:
 - ▶ increment counter atomically
 - ▶ wait (until non-null) and decrement atomically
- ▶ invented by E. Dijkstra
- ▶ operations P (wait-and-dec) and V (increment), also up and down
- ▶ `http://en.wikipedia.org/wiki/Semaphore%5f(programming)`

how to use a semaphore

- ▶ set semaphore to (instances-1) for resource, eg. 0
- ▶ before accessing resource:
- ▶ P(sem): will decrement if $\text{sem} \geq 0$, block otherwise
- ▶ then use resource
- ▶ after use V(sem): will increase sem
- ▶ one other waiter will be unblocked, decrement (atomically) and go

semaphore functions

- ▶ **prototype:**
 - ▶ `int semget (key_t key, int nsems, int semflg);`
 - ▶ `int semctl (int semid, int semnum, int cmd, union semun arg);`
- ▶ get a set of semaphores, init or remove semaphores
- ▶ **prototype:**
 - ▶ `int semop (int semid, struct sembuf *sops, unsigned nsops);`
- ▶ does increment or wait-and-decrement operations

dining philosopher's problem

- ▶ 5 philosophers, round table, 5 forks
- ▶ and (weird) spaghetti that need two forks to be eaten
- ▶ task: get all philosophers fed
- ▶ simple but wrong: wait for left left fork, take it, wait for right and take it, eat, release both.
- ▶ morals: don't lock multiple resources one-by-one
- ▶ <http://en.wikipedia.org/wiki/Dining%5fphilosophers%5fproblem>