CS170: Operating Systems

Week 8 - Discussion Section

Agenda

- Notes on Lab 2
  - Processes
  - wait() and exit()
  - Common Errors
- Introduction to Lab 3
  - dup()
  - dup2()
  - pipe()
  - close()
- Q&A

Processes

- **zombie process** : created when a child process terminates but the parent doesn't call wait()
- **orphan process** : created when a parent process terminates before its child
- **init / sentinel process:** created when the system boots up, responsible for adopting orphaned children

wait() and exit()

- **wait()** : parent process waits for its child process to terminate, then collects its exit status and releases the child's entry from the process table
  - free a zombie child
  - remove child's pid from pid tree / list
  - return child's pid
- **exit()** : process terminates itself, releases its resources, and returns an exit status to the parent process
  - have init / sentinel process adopt all live children
  - if process is not the init / sentinel, free any *zombie* children only
  - if process is the init / sentinel, free *all* children
  - add itself to parent's waiter list
  - set exit code
  - kt_exit()

Common Errors

- Make sure you zero out the registers in execve() before calling PerformExec()
  - when executing argstack, if you see argc=59 and garbage values in argv, this is your issue!
- Make sure you handle PCReg and NextPCReg properly before returning from execve()
  - if you set PCReg=0 and NextPCReg=4, the user code will start at PC=4, which produces nonsensical output
- At the beginning of wait(), before the semaphore call, if you have no waiters and no children, return with an error
  - not doing this can cause weird behavior with ksh
- Make sure that every memory address in your registers is a user-space address
  - therefore, they are offsets from 0

Introduction to Lab 3

- Final feature of KOS: adding a way for processes to communicate with each other
- 4 main tasks:
  - implement file descriptors
  - implement a pipe data structure
  - modify read() and write() to access the pipe data structure
  - implement dup(), dup2(), pipe(), and close()
- Reuses code from Lab 2
- No cookbook :(
- Start early!

dup(int oldfd)

- Duplicate a File Descriptor
  - the new file descriptor actually behaves like an alias of the old one
    - i.e., it refers to the same underlying file as the original file descriptor
  - choose the **lowest numbered file descriptor** that is unopened as the target
- Implementation
  - register 5 stores the given file descriptor
  - make sure the given file descriptor is open (otherwise, return -EBADF)
  - find the lowest free file descriptor in your table
  - copy the fd table entry and increment ref count
  - syscall return new file descriptor on success

dup2(int oldfd, int newfd)

- Duplicate to a Specific File Descriptor
  - similar to with dup(), but **explicitly declare which file descriptor** you want to use as the new one
  - if the new file descriptor is open, close it before duplicating
- Implementation
  - handle arguments:
    - register 5 stores the old file descriptor
    - register 6 stores the new one
    - check their validity
  - make sure the given file descriptor is open (otherwise, return -EBADF)
  - close the new file descriptor if it's already open
  - copy the fd table entry and increment ref count
  - syscall return new file descriptor on success

pipe(int pipefd[2])

- Creates a unidirectional communication channel between two processes
  - contains a write end and a read end
- A **2-element array** to specify the read descriptor and the write descriptor
  - e.g. int pipefd[2]; pipe(pipefd)
- Implementation
  - register 5 stores the pointer
    - check whether it is a valid address
  - find two unused file descriptors
    - if no free file descriptors, syscall return -EMFILE
  - allocate a buffer for the new pipe
  - point the two fd's to the newly allocated buffer
    - pipefd[0] should refer to the read end of the pipe
    - pipefd[1] should refer to the write end of the pipe
    - set ref count
  - syscall return 0 on success

close(int fd)

- If the fd was pointing to the read end of a file:
  - decrement the reader ref_count
  - we don't want writers to block forever on a full pipe, so if this proc was the last active reader: wake up any blocked writers
- If the fd was pointing to the write end of the file:
  - decrement the writer ref_count
  - we don't want readers to block forever on a full pipe, so if this proc was the last active writer: wake up any blocked readers
  - notify readers that they reached eof

Q&A