CS161.A2

Some design considerations.

CS161 2013 A2 Section 1
cs161@eecs.harvard.edu

Max Wang | max.wang@college

Design

- “Implementing to a well-designed interface.”

Design

- "Implementing to a well-designed interface."
- You already know what your components need to do.

Design

- "Implementing to a well-designed interface."
- You already know what your components need to do.
- Design data structures and subroutines that work towards that goal.

File Descriptors

open()

- What happens when we open() a file?

open()

- What happens when we open() a file?
- The user gets an integer.

open()

- What happens when we open() a file?
- The user gets an integer.
- The kernel chooses an index into a *table* of *file descriptors*.

open()

- What happens when we open() a file *twice*?

open()

- What happens when we open() a file *twice*?
- What happens when we `lseek()`?

open()

- What happens when we open() a file *twice*?
- What happens when we `lseek()`?
- Two separate file descriptors

fd

- File descriptors represent *open* files.

fd

- File descriptors represent *open* files.
- What represents a *file*?

vfs

- Virtual file system

vfs

- Virtual file system
- Abstraction layer between file-like systems (filesystem, network, devices) and OS

vfs

- Virtual file system
- Abstraction layer between file-like systems (filesystem, network, devices) and OS
- structs: `vnode`, `uio`

vfs

- Virtual file system
- Abstraction layer between file-like systems (filesystem, network, devices) and OS
- structs: `vnode`, `uio`
- callables: `vfs_*()`, `VOP_*()`

vfs

- Virtual file system
- Abstraction layer between file-like systems (filesystem, network, devices) and OS
- structs: `vnode`, `uio`
- callables: `vfs_*()`, `VOP_*()`
- “All files are the same.”

Plan 9

- “Everything is a file.”

Plan 9

- "Everything is a file."
- Processes are files. (`/proc`)

Plan 9

- "Everything is a file."
- Processes are files. (`/proc`)
- Filetypes using ioctl are just files. (`/net`)

fd table

- What abstraction owns fd tables?

fd table

- What abstraction owns fd tables?
- How do you allocate new fd's (i.e., indices)?

fd table

- What abstraction owns fd tables?
- How do you allocate new fd's (i.e., indices)?
- Should limits be set on the fd table?

fd table

- What abstraction owns fd tables?
- How do you allocate new fd's (i.e., indices)?
- Should limits be set on the fd table?
  - cf. `kern/include/array.h`

fd table

- What abstraction owns fd tables?
- How do you allocate new fd's (i.e., indices)?
- Should limits be set on the fd table?
  - cf. `kern/include/array.h`
- What do you find at `fdtable[i]`?

dup2() / fork()

- What happens when you dup2()...

dup2() / fork()

- What happens when you dup2()...
  - ...and then lseek()?

dup2() / fork()

- What happens when you dup2()...
  - ...and then lseek()?
- What happens when you fork()...

dup2() / fork()

- What happens when you `dup2()`...
  - ...and then `lseek()`?
- What happens when you `fork()`...
  - ...and then `lseek()`?

dup2() / fork()

- What happens when you dup2()...
  - ...and then lseek()?
- What happens when you fork()...
  - ...and then lseek()?
  - ...and then open()?

close()

- A file descriptor may be multiply referenced

`close()`

- A file descriptor may be multiply referenced
- What happens if we `close()` a `dup2()`'d file descriptor?

`close()`

- A file descriptor may be multiply referenced
- What happens if we `close()` a `dup2()`'d file descriptor?
- Use pointers and reference counting.

Processes

proc

- A process is memory + execution state.

proc

- A process is memory + execution state.
  - i.e., address space + thread(s)

proc

- A process is memory + execution state.
  - i.e., address space + thread(s)
- Even though you're single-threading, think about the separation.

proc

- A process is memory + execution state.
  - i.e., address space + thread(s)
- Even though you're single-threading, think about the separation.
- (Nobody says you *can't* try multi-threading...)

proc

proc

- What per-process state might we want?

proc

- What per-process state might we want?
  - pid, address space, fd table

proc

- What per-process state might we want?
  - pid, address space, fd table
- What about per-thread state?

proc

- What per-process state might we want?
  - pid, address space, fd table
- What about per-thread state?
  - cwd, execution state, user stack pointer

pid

- As with fd's, think about...

pid

- As with fd's, think about...
- ...how to allocate/free pids.

pid

- As with fd's, think about...
- ...how to allocate/free pids.
- ...whether to enforce a limit.

pid

- As with fd's, think about...
- ...how to allocate/free pids.
- ...whether to enforce a limit.
- ...modularity.

exit()/waitpid()

- Let's free up procs on `exit()`.

exit()/waitpid()

- Let's free up procs on `exit()`.
- What if someone is waiting on our exit status?

`exit()` / `waitpid()`

- Let's free up procs on `exit()`.
- What if someone is waiting on our exit status?
- Who can wait on our exit status?

`exit()` / `waitpid()`

- Let's free up procs on `exit()`.
- What if someone is waiting on our exit status?
- Who can wait on our exit status?
- Parent/child process hierarchy

exit() / waitpid()

- Let's free up child procs on `waitpid()`.

`exit()` / `waitpid()`

- Let's free up child procs on `waitpid()`.
- What if our parent exits?

`exit()` / `waitpid()`

- Let's free up child procs on `waitpid()`.
- What if our parent exits?
- Don't want orphan zombies

`exit()` / `waitpid()`

- Let's free up child procs on `waitpid()`.
- What if our parent exits?
- Don't want orphan zombies
- (What is a zombie?)

`exit()` / `waitpid()`

- Let's free up child procs on `waitpid()`.
- What if our parent exits?
  - Don't want orphan zombies
  - (What is a zombie?)
- `init`: ancestor of all processes

exit()/waitpid()

- Don't forget to synchronize! Think about how you...

`exit()` / `waitpid()`

- Don't forget to synchronize! Think about how you...
- ...wait for an `exit()`

`exit()` / `waitpid()`

- Don't forget to synchronize! Think about how you...
- ...wait for an `exit()`
- ...implement `WNOHANG`

`exit()` / `waitpid()`

- Don't forget to synchronize! Think about how you...
- ...wait for an `exit()`
- ...implement `WNOHANG`
- ...prevent deadlock

`exit()` / `waitpid()`

- Don't forget to synchronize! Think about how you...
  - ...wait for an `exit()`
  - ...implement `WNOHANG`
  - ...prevent deadlock
  - ...maintain process hierarchy w/o race

`fork()`

- Create a new process *and* thread, identical to the caller.

`fork()`

- Create a new process *and* thread, identical to the caller.
- Need to copy process stuff

fork()

- Create a new process *and* thread, identical to the caller.
- Need to copy process stuff
  - address space, fd table, pid

`fork()`

- Create a new process *and* thread, identical to the caller.
- Need to copy process stuff
  - address space, fd table, pid
  - lol j/k—allocate a new pid

fork()

- Need to copy thread stuff also!

fork()

- Need to copy thread stuff also!
  - cwd, execution state

fork()

- Need to copy thread stuff also!
  - cwd, execution state
  - The *trapframe* contains execution state

fork()

- Need to copy thread stuff also!
  - cwd, execution state
  - The *trapframe* contains execution state
    - e.g., PC and return value

fork()

- Need to copy thread stuff also!
  - cwd, execution state
  - The *trapframe* contains execution state
    - e.g., PC and return value
  - Set up process hierarchy before returning

execv()

- Replace a process's address space and execution state with a new binary

execv()

- Replace a process's address space and execution state with a new binary
- `runprogram()`

execv()

- Replace a process's address space and execution state with a new binary

execv()

- Replace a process's address space and execution state with a new binary
- `runprogram()`

execv()

- What does `runprogram()` do?

execv()

- What does `runprogram()` do?
  - Open a binary file

execv()

- What does `runprogram()` do?
  - Open a binary file
  - Create a new address space

execv()

- What does `runprogram()` do?
  - Open a binary file
  - Create a new address space
  - Load the executable there

execv()

- What does `runprogram()` do?
  - Open a binary file
  - Create a new address space
  - Load the executable there
  - Define the stack

execv()

- What does `runprogram()` do?
  - Open a binary file
  - Create a new address space
  - Load the executable there
  - Define the stack
  - Enter usermode

execv()

- What doesn't `runprogram()` do?

execv()

- What doesn't `runprogram()` do?
  - argument handling

execv()

- What doesn't `runprogram()` do?
  - argument handling
  - handling arguments

execv()

- What doesn't `runprogram()` do?
  - argument handling
  - handling arguments
  - putting argv where argv has to go

execv()

- What doesn't `runprogram()` do?
  - argument handling
  - handling arguments
  - putting argv where argv has to go
  - copying arguments into your new addrspace

Error handling

- Error handling is crucial and often hard
- What if you free your old address space, but you fail later in `execv()`?
- What if you fork a thread but the new thread dies before the fork finishes?

Scheduling

Scheduler

- Your scheduler should be swappable with round-robin.
  - Use #ifdef's to achieve this.
- Your scheduler needs to take multiprocessing into account!

MLFQs

- Multi-level feedback queues

MLFQs

- Multi-level feedback queues
- Higher priority == shorter timeslice

MLFQs

- Multi-level feedback queues
- Higher priority == shorter timeslice
- Timeslices increase exponentially against priority.

MLFQs

- Multi-level feedback queues
- Higher priority == shorter timeslice
- Timeslices increase exponentially against priority.
- Demote a thread if it uses its entire timeslice; promote it otherwise.

Random

- Choose a random thread and run it.

Random

- Choose a random thread and run it.
- Works surprisingly well!

Random

- Choose a random thread and run it.
- Works surprisingly well!
- Fair, simple, low overhead, no starvation

Lottery

- Assign a variable number of tickets to each thread.

Lottery

- Assign a variable number of tickets to each thread.
- Choose a random ticket and run its holder.

Lottery

- Assign a variable number of tickets to each thread.
- Choose a random ticket and run its holder.
- Flexible, no starvation (if min 1 ticket)

Scheduler

- Do something better than round-robin.
- Convince us about why it's good.