Intro to lab 2: time-shared multiprogramming

First task: allow the user to compile/run programs that use malloc() + standard I/O library

- ioctl() (kinda)
- fstat() (kinda)
- sbrk()
- ...

Second task: implement system calls for a simple shell

- fork()
- execve()
- wait()
- exit()

Third task: implement process IDs

- getpid()
- getppid()

Fourth task: make time slicing work

The sbrk(), fstat(), and ioctl() system calls

*void *sbrk(intptr_t increment);*
*int fstat(int fd, struct stat *statbuf);*
int ioctl(int fd, unsigned long request, ...);

- sbrk() increments the location of the **program break**
  - The division between the heap and unallocated memory
  - Return the location of the program break **before** incrementing (yeah it's weird...)
- We only implement one case of ioctl() for this lab
  - Use **ioctl_console_fill()**
- For fstat(), use **stat_buf_fill()**
  - We only care about file descriptors 0, 1, and 2
    - fd 0 → size = 1
    - fd 1, 2 → size = 256
- The man pages are your best friend!

Some other system calls

- close()
  - Return (-1 * EBADF)
- getdtablesize()
  - Return the value 64
- getpagesize()
  - Return the value of PageSize in simulator.h (512)

The fork(), execve(), and wait() system calls

pid_t fork(void);
*pid_t wait(int *_Nullable wstatus);*
*int execve(const char *pathname, char *const _Nullable argv[], char *const _Nullable envp[]);*

- wait() — when called by a parent, waits until a child exits
  - After the child exits, clean up its PCB
  - Add two fields to your PCB data structure
    - unsigned short pid
    - Pointer to parent PCB
- execve() — malloc some space on the heap to save pathname and argv[]
  - When you call load_user_program(), you'll overwrite everything
  - Ignore envp[] — we don't have environment variables
- fork() — duplicate the current process (except for the PIDs)
  - Child gets 0
  - Parent gets child's PID
  - Queue both onto the scheduler

Child exits:

- Clean up its own stuff (deallocate mem partition)
- V(semaphore)

Parent waits:

- P(semaphore)
- Clean up child
    - Free the PCB
    - Delete the dll node

Task 3: implementing process IDs

pid_t getpid(void);
pid_t getppid(void);

- Every process needs a unique ID
  - You can use a red-black tree (or not!)
  - /cs/faculty/rich/cs170/include/jrb.h

Task 4: time slicing

- This lets us switch between processes!
- Just call start_timer(ticks) once
  - I did it at the end of InitUserProcess()
  - You can use ticks = 10 (or whatever, really)
- Simulator throws a TimerInt every time interval
  - When handling this, just put the current running process at the end of your ready queue