Assignment 2

Adding a user program to MavOS

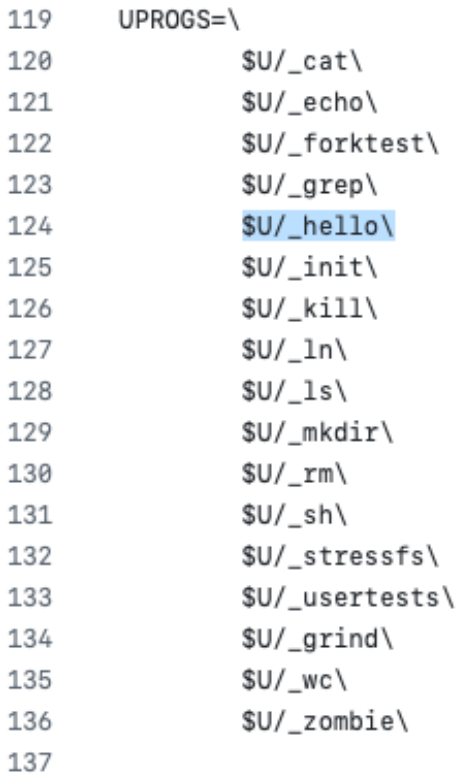
hello.c

```
#include "types.h"
#include "stat.h"
#include "user.h"


int main(int argc, char *argv[])
{
  printf(“Hello, World\n”);
  exit();
}
```

- The **`user`** directory is where all user space code and user applications live.

- Create your new program .c file in the user directory.

Add the user program to the Makefile



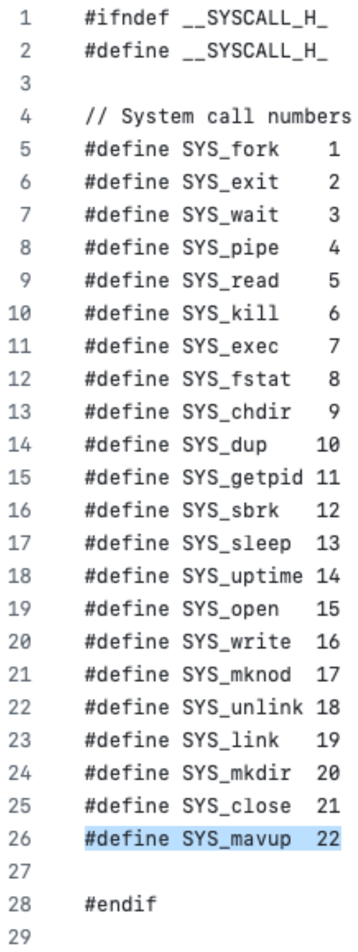
```
UPROGS=\
	$U/_cat\
	$U/_echo\
	$U/_forktest\
	$U/_grep\
	$U/_hello\
	$U/_init\
	$U/_kill\
	$U/_ln\
	$U/_ls\
	$U/_mkdir\
	$U/_rm\
	$U/_sh\
	$U/_stressfs\
	$U/_usertests\
	$U/_grind\
	$U/_wc\
	$U/_zombie\

```

- MacOS/Makefile contains the make rules to build our OS.
- In the UPROGS definition add a line for your new program.
    - Note you need to prepend _ to the filename.

Steps to add a system call

- You will modify four files to add a new system call
    1. `syscall.h` - Reserve a system call number for your system call.
    2. `syscall.c` - Add an entry in the system call vector for your system call.
    3. `sysproc.c` - Add the system call function
    4. `usys.S` - Add a SYSCALL entry to allow the system to generate an entry point for your system call
    5. `user.h` - Add a function signature to the user space header file

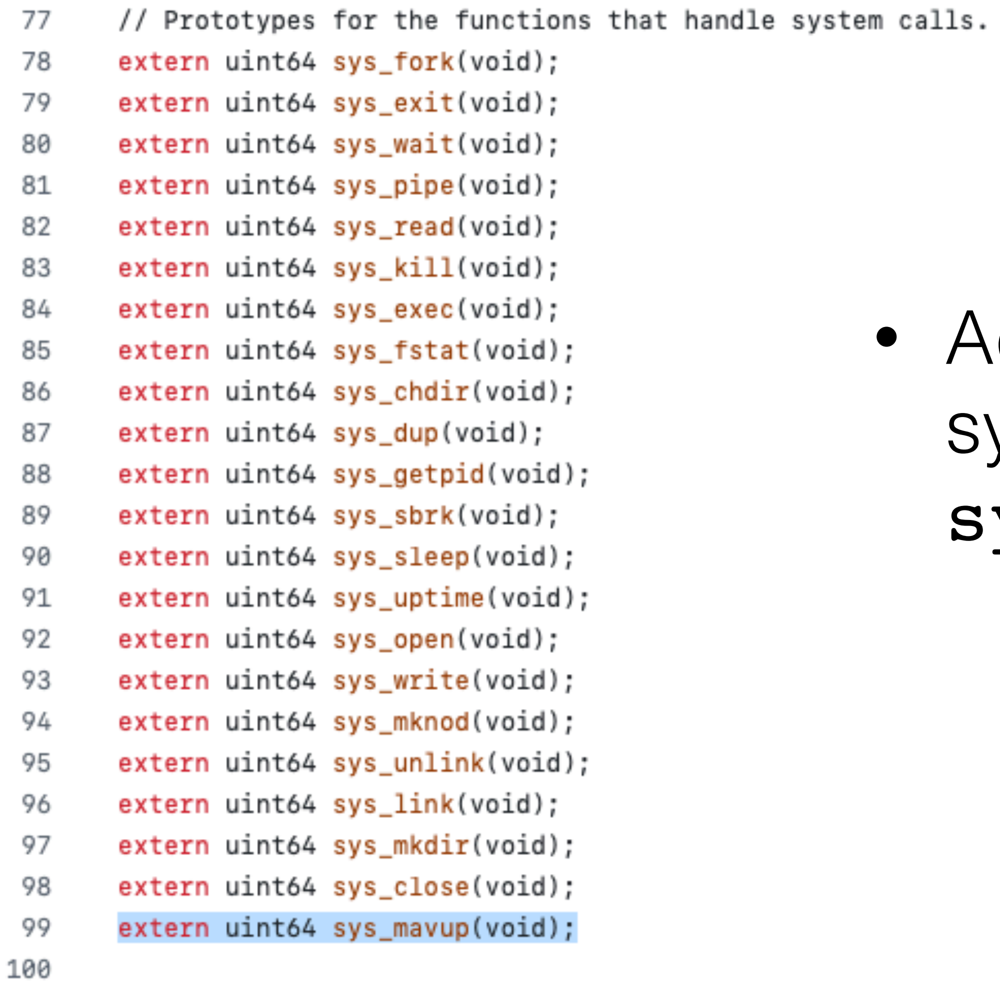
Reserving a system call number



```
#ifndef __SYSCALL_H_
#define __SYSCALL_H_

// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_mavup  22

#endif
```

- Claim the next integer for your system call by adding a new `#define` in `syscall.h`

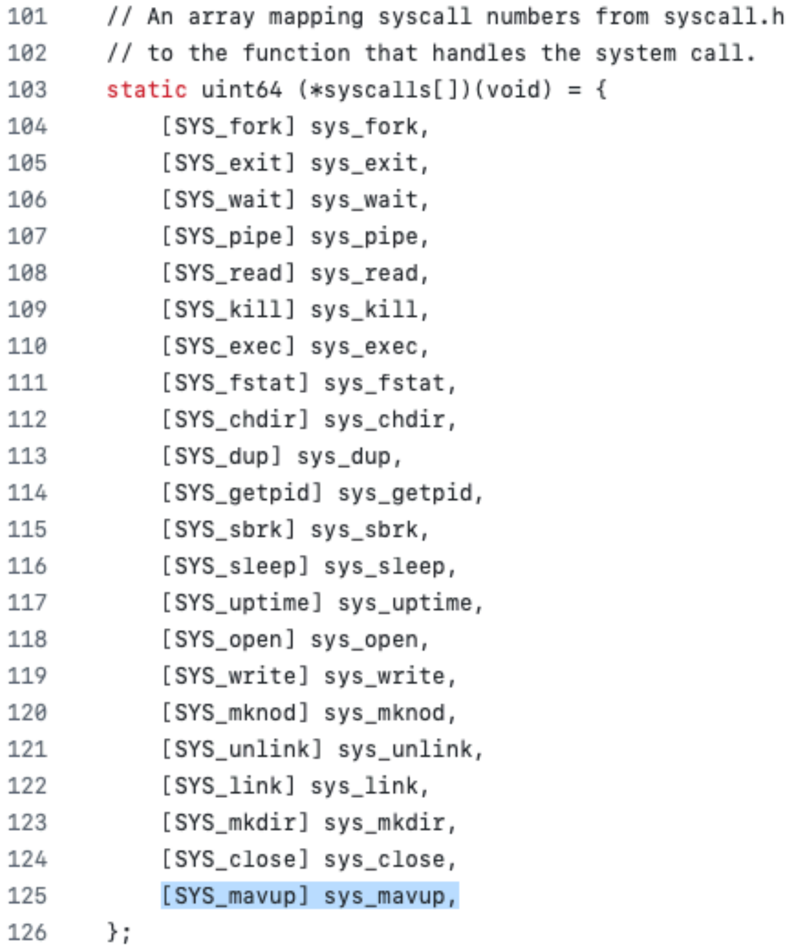
Add prototype for your system call



```
// Prototypes for the functions that handle system calls.
extern uint64 sys_fork(void);
extern uint64 sys_exit(void);
extern uint64 sys_wait(void);
extern uint64 sys_pipe(void);
extern uint64 sys_read(void);
extern uint64 sys_kill(void);
extern uint64 sys_exec(void);
extern uint64 sys_fstat(void);
extern uint64 sys_chdir(void);
extern uint64 sys_dup(void);
extern uint64 sys_getpid(void);
extern uint64 sys_sbrk(void);
extern uint64 sys_sleep(void);
extern uint64 sys_uptime(void);
extern uint64 sys_open(void);
extern uint64 sys_write(void);
extern uint64 sys_mknod(void);
extern uint64 sys_unlink(void);
extern uint64 sys_link(void);
extern uint64 sys_mkdir(void);
extern uint64 sys_close(void);
extern uint64 sys_mavup(void);

```

- Add a prototype for your system call function in **`syscall.c`**

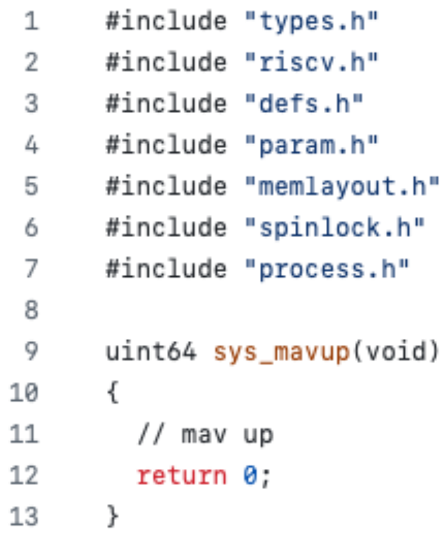
Add entry for your system call to the syscall vector



```
// An array mapping syscall numbers from syscall.h
// to the function that handles the system call.
static uint64 (*syscalls[])(void) = {
    [SYS_fork] sys_fork,
    [SYS_exit] sys_exit,
    [SYS_wait] sys_wait,
    [SYS_pipe] sys_pipe,
    [SYS_read] sys_read,
    [SYS_kill] sys_kill,
    [SYS_exec] sys_exec,
    [SYS_fstat] sys_fstat,
    [SYS_chdir] sys_chdir,
    [SYS_dup] sys_dup,
    [SYS_getpid] sys_getpid,
    [SYS_sbrk] sys_sbrk,
    [SYS_sleep] sys_sleep,
    [SYS_uptime] sys_uptime,
    [SYS_open] sys_open,
    [SYS_write] sys_write,
    [SYS_mknod] sys_mknod,
    [SYS_unlink] sys_unlink,
    [SYS_link] sys_link,
    [SYS_mkdir] sys_mkdir,
    [SYS_close] sys_close,
    [SYS_mavup] sys_mavup,
};
```

Add an entry for your system call function in the sys call vector

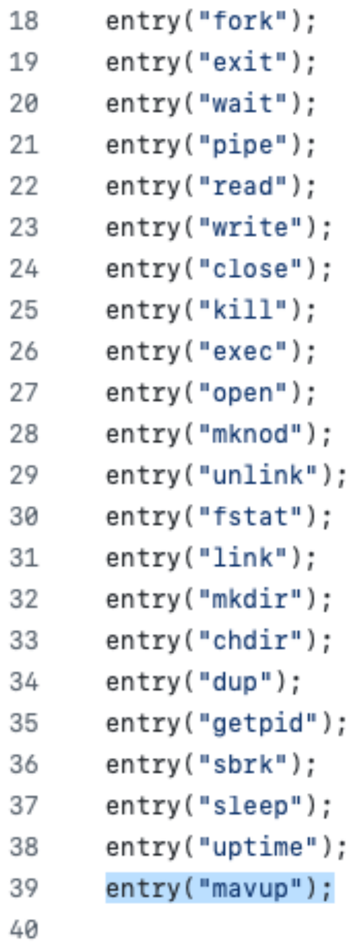
Implement your system call function



```
#include "types.h"
#include "riscv.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "spinlock.h"
#include "process.h"

uint64 sys_mavup(void)
{
  // mav up
  return 0;
}
```

- In `sysproc.c` add the code for your system call function

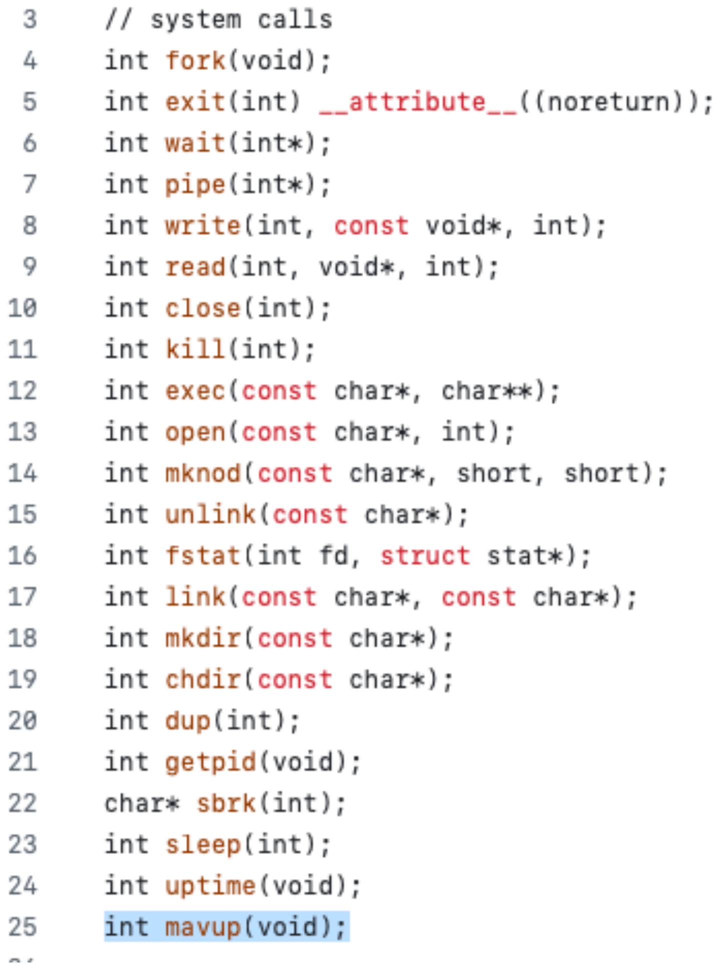
Now create the user space hooks



```
entry("fork");
entry("exit");
entry("wait");
entry("pipe");
entry("read");
entry("write");
entry("close");
entry("kill");
entry("exec");
entry("open");
entry("mknod");
entry("unlink");
entry("fstat");
entry("link");
entry("mkdir");
entry("chdir");
entry("dup");
entry("getpid");
entry("sbrk");
entry("sleep");
entry("uptime");
entry("mavup");
```

- In **`user/usys.pl`** add an entry for your system call.

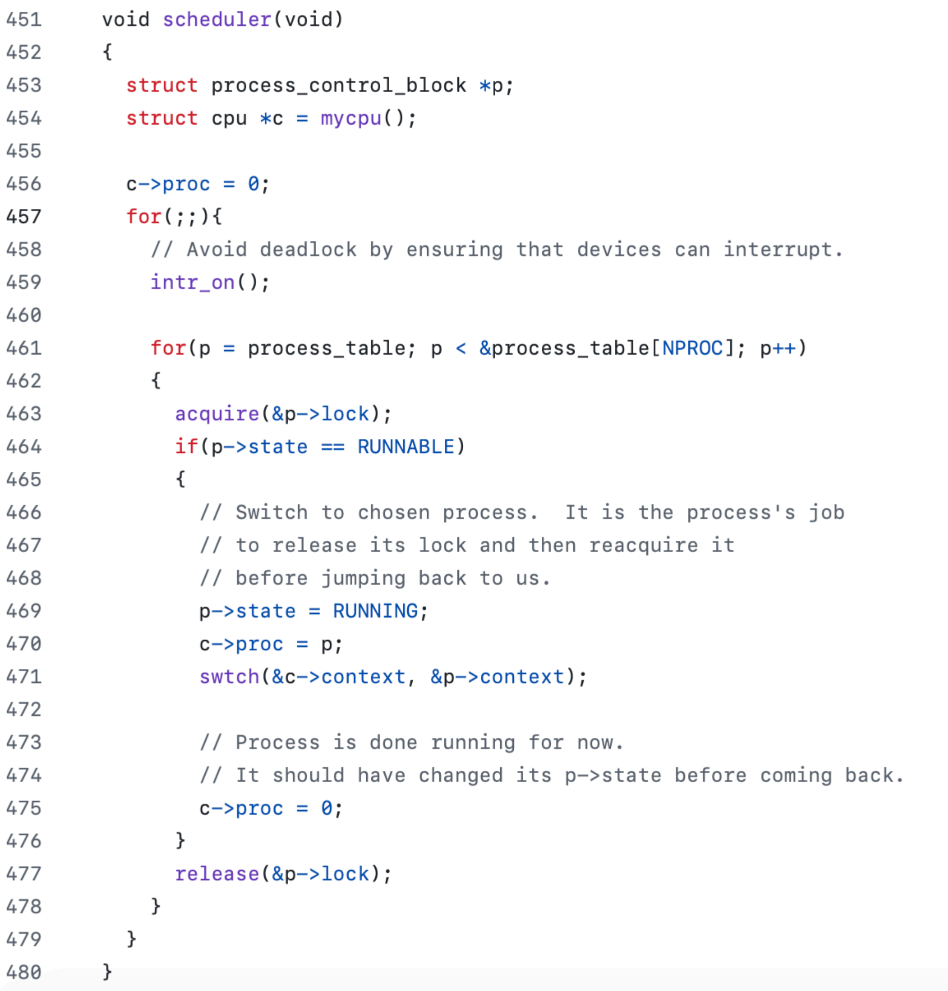
Add a function prototype in the user space header



```
// system calls
int fork(void);
int exit(int) __attribute__((noreturn));
int wait(int*);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(const char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int mavup(void);
```

- Add the function prototype to `user/user.h`

Scheduler



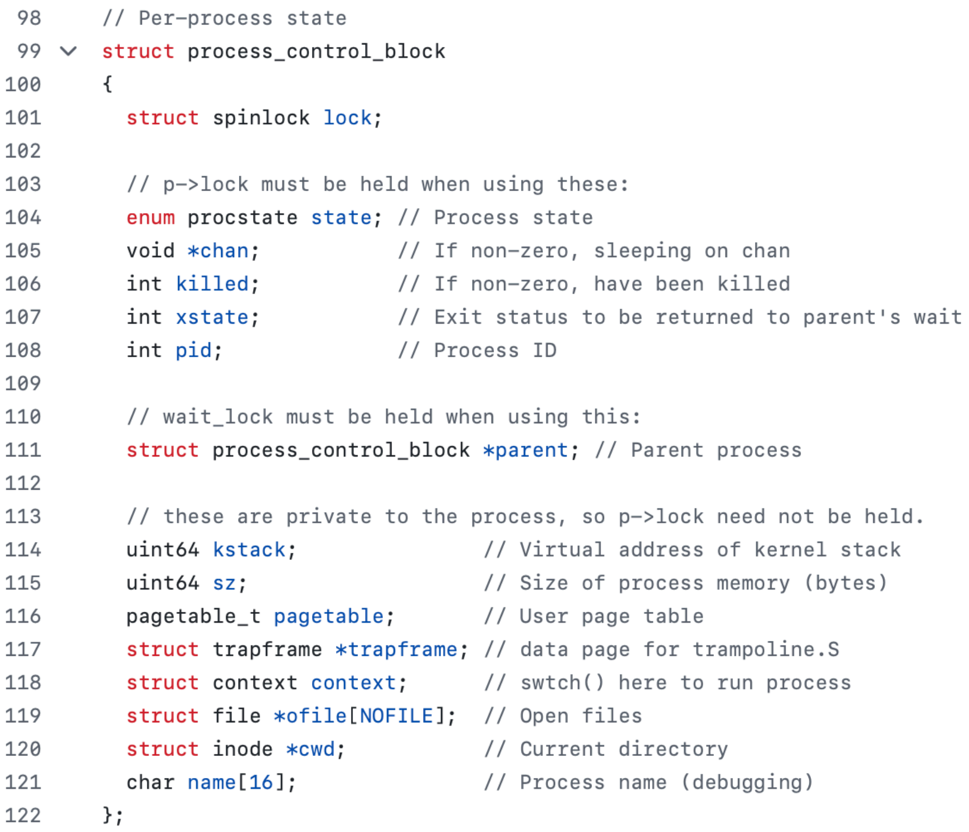
```
void scheduler(void)
{
  struct process_control_block *p;
  struct cpu *c = mycpu();

  c->proc = 0;
  for(;;){
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();

    for(p = process_table; p < &process_table[NPROC]; p++)
    {
      acquire(&p->lock);
      if(p->state == RUNNABLE)
      {
        // Switch to chosen process.  It is the process's job
        // to release its lock and then reacquire it
        // before jumping back to us.
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);

        // Process is done running for now.
        // It should have changed its p->state before coming back.
        c->proc = 0;
      }
      release(&p->lock);
    }
  }
}
```

- `process.c` contains the scheduler function `scheduler()`.

- The current scheduler implements round robin.

  - Change the for loop to choose based on priority rules

process control block

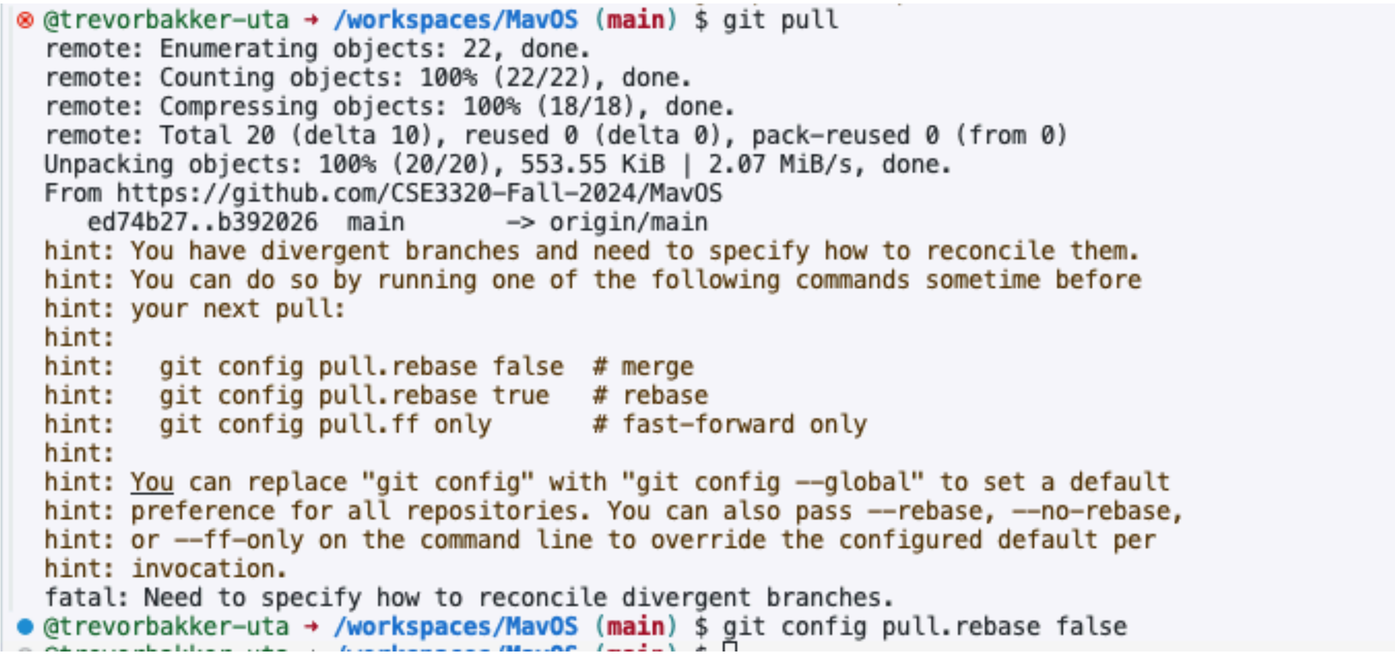


```
// Per-process state
struct process_control_block
{
  struct spinlock lock;

  // p->lock must be held when using these:
  enum procstate state; // Process state
  void *chan;           // If non-zero, sleeping on chan
  int killed;           // If non-zero, have been killed
  int xstate;           // Exit status to be returned to parent's wait
  int pid;              // Process ID

  // wait_lock must be held when using this:
  struct process_control_block *parent; // Parent process

  // these are private to the process, so p->lock need not be held.
  uint64 kstack;               // Virtual address of kernel stack
  uint64 sz;                   // Size of process memory (bytes)
  pagetable_t pagetable;       // User page table
  struct trapframe *trapframe; // data page for trampoline.S
  struct context context;      // swtch() here to run process
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

- The process control block is declared in `process.h`

- `allocproc(void)` allocates a new process and is a good spot to initialize the new process control block members.

Common git message



```
⊗ @trevorbakker-uta → /workspaces/MavOS (main) $ git pull
  remote: Enumerating objects: 22, done.
  remote: Counting objects: 100% (22/22), done.
  remote: Compressing objects: 100% (18/18), done.
  remote: Total 20 (delta 10), reused 0 (delta 0), pack-reused 0 (from 0)
  Unpacking objects: 100% (20/20), 553.55 KiB | 2.07 MiB/s, done.
  From https://github.com/CSE3320-Fall-2024/MavOS
     ed74b27..b392026  main       -> origin/main
  hint: You have divergent branches and need to specify how to reconcile them.
  hint: You can do so by running one of the following commands sometime before
  hint: your next pull:
  hint:
  hint:   git config pull.rebase false  # merge
  hint:   git config pull.rebase true   # rebase
  hint:   git config pull.ff only       # fast-forward only
  hint:
  hint: You can replace "git config" with "git config --global" to set a default
  hint: preference for all repositories. You can also pass --rebase, --no-rebase,
  hint: or --ff-only on the command line to override the configured default per
  hint: invocation.
  fatal: Need to specify how to reconcile divergent branches.
● @trevorbakker-uta → /workspaces/MavOS (main) $ git config pull.rebase false
```

Type: `git config pull.rebase false`