csc/cpe 357 Lateterm

Spring 2021

Name: Brenden Sprague

User ID (email): baspragu@calpoly.edu

Rules:

- Do all your own work. Nothing says your neighbor has any better idea what the answer is. Plus, this quarter working from home, you don't have a neighbor.
- This exam is open book, notes, internet, and anything inanimate.
- If you unsure if a resource is animate, ask it. If it answers, it is.
- Do not discuss this exam outside of class until after 11:59pm, Monday, May 24th.
- If you need to add a picture or any other "extra" thing, put a note in the text box and submit your picture via handin along with the exam.
- Submit this exam via `handin` to `lateterm` by 23:59 tonight. (I'm not expecting you to spend all day on this, but you get to chose when.)
- The programming problems should be submitted in the specified files.
- As insurance, you may wish to include plain text versions of the short-answer problems, too.
- This exam is copyright © 2021 Phillip Nico. Unauthorized redistribution is prohibited.

Suggestions(mostly the obvious):

- When in doubt, state any assumptions you make in solving a problem. **If you think there is a misprint, ask me.**
- **Read the questions carefully.** Be sure to answer all parts.
- Identify your answers *clearly.*
- Watch the time/point tradeoff: 50pts / 50 min works out to 60.0s/pt.
- Problems are not necessarily in order of difficulty. They are in the order in which they fit.
- Be sure you have all pages. Pages other than this one are numbered "n of 8".

Encouragement:

- Good Luck!

Problem	Possible	Score
1	5	
2	5	
3	10	
4	10	
5	20	
Total:	**50**	

```
   I know the answer!  The answer lies within the heart of all mankind!  The answer is twelve?  I think I'm in
the wrong building.
   -- Charles Schulz (as quoted by /usr/games/fortune)
```

Answer clearly, concisely, and (where possible) correctly:

1. (5) What is the fundamental difference between a system call and a library function?

   There are a few fundamental differences:

   (1) Library functions are linked with the users written code into a single process, which have the same privileges. System calls however make special calls from the users program, to the kernel, which can access privileges if necessary.

   [EXAMPLE] System calls acts as entry point to OS kernel, to accomplish things like changing system characteristics such as date/time, creating sockets and utilizing network protocols, or interacting with hardware using signals.

   (2) Library functions mostly do not have to worry about permissions as they are running with the processes in the user space on memory, but system calls work in the kernel space, therefor requiring an interrupt routines and jumps to get to that space. Since system calls have access to everything in the system, there is more control needed under the hood to make sure the system call is correctly handling permissions, for example. correctly. Due to these two fundamental differences, library functions are FASTER than system calls.

2. (5) If a user with a umask of `0527` attempts to create a file with the call:

   ```
   open("examfile", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
   ```

   Given that `examfile` does not exist, what will permissions of the created file be?

   --w-r-----
   (240)

3. (10) Write a C function called `bstr2int()` that takes a valid C string (possibly null) consisting exclusively of the digits "0" and "1", strips off any leading zeros, and then builds an int out of the remaining values. Returns the resulting integer on success or -1 if the number cannot be represented (due to overflow). Recall that the size of an int on the current platform can be determined through `sizeof(int)`.

   For example, `bstr2int("00001100")` would return 12, while `bstr2int(null)` would return 0 (no bits set). Write robust code.

   This is the space I would've given if this were an in-person exam. Submit your code as `p3.c`.

   ```
   int bstr2int(const char *s) {
   ```

   }

4. (10) The UNIX system utility `cp` will refuse to copy a file if the source and destination files are the same. (how does it know?) No, really, how does it know?

   Write a c function called `same_file` that takes two pathnames and returns true (nonzero) if both paths specify the same file and false otherwise (if either one doesn't exist (or is inaccessible), or they are not the same).

   This is the space I would've given if this were an in-person exam. Submit your code as `p4.c`.

```
int same_file(const char *src, const char *dst) {

}
```

5. (20) Write a function `myfind()` that takes two arguments, a path to a directory and a name, and prints out all filesystem objects (files, directories, whatever) with that name under that directory. It returns the number of objects found. A demo version of a program using this can be found in `~pn-cs357/demos/myfind` if you want to play with it..

   notes:
   - You may use `PATH_MAX` as defined in `limits.h` and abandon any paths longer than that.
   - Do not follow symbolic links, nor should you follow "." or ".." lest you recurse infinitely.
   - If you encouter errors (unreadable directories, etc.) report them, but do not terminate execution.
   - Don't waste file descriptors, but you may assume you have however many you need for one per level for the deepest tree. (there's a hint in this; I shan't explain.)
   - Remember, this is different from most exam coding experiences. you have a unix development environment. take advantage of it.
   - To be explicit, if the last link of the given path (path) matches the given name (name), this is **not** a hit. (Whether or not that's the right decision, it makes it easier.)
   - Write robust code.

   example:
   ```
   % myfind ./midterm makefile
   ./midterm/makefile
   ./midterm/play/myfind/makefile
   ```

   This is the space I would've given if this were an in-person exam. Submit your code as `p5.c`.

   ```
   int myfind(const char *path, const char *name) {
   ```

(Continued on the following page. . .)

Optional extra space for problem 5.

}

Useful Information

Prototypes

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fildes, off_t offset, int whence);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
int close(int fd);
int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
int symlink(const char *oldpath, const char *newpath);
int readlink(const char *path, char *buf, size_t bufsiz);
int link(const char *oldpath, const char *newpath);
int unlink(const char *pathname);
int utime(const char *filename, struct utimbuf *buf);
int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
char *getcwd(char *buf, size_t size);
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
int chdir(const char *path);
int fchdir(int fd);
DIR *opendir(const char *name);
void rewinddir(DIR *dir);
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);
off_t telldir(DIR *dir);
struct passwd *getpwnam(const char * name);
struct passwd *getpwuid(uid_t uid);
struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg , ...,
    char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *filename, char *const argv [],
    char *const envp[]);
int kill(pid_t pid, int sig);
pid_t fork(void);
pid_t wait(int *status)
pid_t waitpid(pid_t pid, int *status, int options);
int sigaction(int signum, const struct sigaction *act,
    struct sigaction *oldact);
int sigprocmask(int how, const sigset_t *set,
    sigset_t *oldset);
int sigpending(sigset_t *set);
int sigsuspend(const sigset_t *mask);
void (*signal(int signum, void (*sighandler)(int)))(int);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
int pipe(int filedes[2]);
unsigned int alarm(unsigned int seconds);
int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value,
    struct itimerval *ovalue);
int tcgetattr ( int fd, struct termios *termios_p );
int tcsetattr ( int fd, int optional_actions, struct
    termios *termios_p );
int feof( FILE *stream);
int ferror( FILE *stream);
int fileno( FILE *stream);
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
```

Structures and Macros

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}



struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
};

struct timeval {
    long tv_sec;
    long tv_usec;
};

ITIMER_REAL
ITIMER_VIRTUAL
ITIMER_PROF



struct passwd {
        char    *pw_name;
        char    *pw_passwd;
        uid_t   pw_uid;
        gid_t   pw_gid;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};



struct group {
        char    *gr_name;
        char    *gr_passwd;
        gid_t   gr_gid;
        char    **gr_mem;
};



struct stat
{
    dev_t         st_dev;
    ino_t         st_ino;
    mode_t        st_mode;
    nlink_t       st_nlink;
    uid_t         st_uid;
    gid_t         st_gid;
    dev_t         st_rdev;
    off_t         st_size;
    unsigned long st_blksize;
    unsigned long st_blocks;
    time_t        st_atime;
    time_t        st_mtime;
    time_t        st_ctime;
};
```

Structures and Macros, cont.

```
S_ISLNK(m)   S_ISDIR(m)   S_ISBLK(m)
S_ISREG(m)   S_ISCHR(m)   S_ISFIFO(m)
                          S_ISSOCK(m)
```

Macro	Value	Macro	Value
S_IFMT	0170000	S_IRWXU	00700
S_IFSOCK	0140000	S_IRUSR	00400
S_IFLNK	0120000	S_IWUSR	00200
S_IFREG	0100000	S_IXUSR	00100
S_IFBLK	0060000	S_IRWXG	00070
S_IFDIR	0040000	S_IRGRP	00040
S_IFCHR	0020000	S_IWGRP	00020
S_IFIFO	0010000	S_IXGRP	00010
S_ISUID	0004000	S_IRWXO	00007
S_ISGID	0002000	S_IROTH	00004
S_ISVTX	0001000	S_IWOTH	00002
		S_IXOTH	00001

```
struct dirent
{
    long d_ino;
    off_t d_off;
    unsigned short d_reclen;
    char d_name [NAME_MAX+1];
}
```

```
struct utimbuf {
        time_t actime;
        time_t modtime;
};
```

```
struct termios {
    tcflag_t c_iflag;
    tcflag_t c_oflag;
    tcflag_t c_cflag;
    tcflag_t c_lflag;
    cc_t c_cc[NCCS];
}
```

```
WIFEXITED(status)
WEXITSTATUS(status)
WIFSIGNALED(status)
WTERMSIG(status)
WIFSTOPPED(status)
WSTOPSIG(status)
```