# CSCI 4061: Finale

Chris Kauffman

*Last Updated:*
*Thu May 2 17:19:31 CDT 2019*

## Logistics

Lab 14

- 50% Attendance
- 50% Online Exit Survey

P2: Due Mon 5/6

- Will accept up to 2 days late
- ADVANCED feature tests posted later today

Questions?

Today: Review, Course Evals

| Date | Event |
|---|---|
| Thu 4/25 | Sockets Basics |
| Mon 4/29 | Sockets Lab |
| Tue 4/30 | Sockets Wrap |
| Thu 5/2 | Review |
| Mon 5/6 | Review Lab |
| | A2 Due |
| Mon 5/13 | **8-10am Final Exam** |

## What have we done?

### Unix Systems Programming

API of Unix system for files, processes, signals, IPC, threads, sockets, memory

### Glimpses of OS Internals

Process accounting, file representation, communication buffers

### Concurrency and Communication

Protocols to allow distinct operators to cooperate/communicate without deadlocking

### C Programming

Memory allocation, pointers, structs, conventions for errors

### Did I miss anything?

## Further Reading

- INTERNALS: The Design of the UNIX Operating System by Maurice A. Bach : Step-by-step treatment of the original design internals of the Unix OS. Lots of pictures and great discussion of concurrency issues in the kernel.
- DESIGN: The Art of Unix Programming by Eric S. Raymond : Fantastic philosophical and pragmatic discussion of how to build systems that work especially in the Unix environment. (free online)
- HUMANS: Coders at Work: Reflections on the Craft of Programming by Peter Seibel : Fascinating interviews with notable programmers who got the job done including AI giant Peter Norvig, Scheme Inventor Guy Steele, original Unix inventor Ken Thompson, and CS godfather Donald Knuth.

## Final Exam

### Logistics

- Mon 5/13 8:00am-10:am in this room
- 5-6 sides of paper, write on them, hand in
  - Midterms were 4 sides of paper
- No bluebook or bubble sheet required
- Comprehensive, combination of coding, analysis, short answer
- Open Resource as were the midterm exams

### Topics Request

Any particular topics folks would like to discuss prior to review questions?

## Review Question 1

A binary file stores many `mesg_t` structs in it; these structs have the definition:

```
typedef struct {
  int kind;
  char name[256];
  char body[1024];
} mesg_t;
```

Define the following function

```
int print_all(char *filename, char *user_name)
// Open the given filename which stores binary mesg_t structs scan the
// file for mesg_t's with a name field that matches the given
// user_name and print their bodies. Close the file and return the
// number of messages found for the given the user_name.
```

**Bonus point:** provide a version that uses uses the `mmap()` function rather than standard I/O functions.

## Answers:

```
int print_all(char *filename,
              char *user_name)
{
  int fd = open(filename, O_RDONLY);
  mesg_t msg;
  int count = 0;
  while(1){
    int nbytes = read(fd, &msg,
                      sizeof(mesg_t));
    if(nbytes==0){
      break;
    }
    if( strcmp(msg.name, user_name) == 0 )
    {
      printf("%s\n",msg.body);
      count++;
    }
  }
  close(fd);
  return count;
}
```

```
int print_all(char *filename,
              char *user_name)
{
  int fd = open(filename, O_RDONLY);
  struct stat statbuf;
  fstat(fd, &statbuf );
  int len = statbuf.size / sizeof(mesg_t);
  mesg_t *mesgs =
    mmap(NULL,statbuf.size,
         PROT_READ, MAP_PRIVATE,
         fd, statbuf.size);
  int count = 0;
  for(int i=0; i<len; i++){
    if(strcmp(mesgs[i].name, user_name)==0)
    {
      printf("%s\n",mesgs[i].body);
      count++;
    }
  }
  munmap(mesgs);
  close(fd);
  return count;
}
```

## Review Question 2

The blather server `bl_server` was required to use the `select()` system call to check whether its various input sources were ready. The general pattern was as follows.

```
repeat {
  use select() to check join
  and client FIFOs
  if join is ready{
    read a join request
    and process it
  }
  for each client C{
    if client C is ready{
      read a message from C
      and process it
    }
  }
}
```

A much simpler pattern of I/O would not use `select()` such as the below.

```
repeat {
  read a join request
  and process it
  for each client C {
    read a message from C
    and process it
  }
}
```

**Discuss the differences** in behavior between these two and any undesirable outcomes if the second pattern were used.

## Answers:

Select is used to check all possible input sources to discover which is ready for immediate reading. In the alternate version, the top of each server loop will `read()` from the join FIFO. At first this may seem to work as a client will be able to join. However, if a client joins successfully, it will likely send a message which will not be immediately accepted by the server. This is likely due to the server again `read()`'ing from the join FIFO which blocks until another client joins. Only then will the server enter the loop to check for client inputs. `select()` avoids this problem by blocking only when no input sources are available and returning immediately when any source is ready.

## Review Problem 3

`bl_server` uses `select()` to detect which input sources are ready while `bl_client` uses multiple threads to handle its input sources. **Discuss** using multiple threads in `bl_server` instead of `select()` to handle its various input sources. In your answer describe the following:

- How many threads will be required for the server
- When threads will be created and ended (canceled)
- What would each server thread DO (deal with joining, deal with one client or multiple clients, etc.)
- What kind of coordination needs to exist between server threads to facilitate broadcast operations (writing to multiple client output)
- What kind of coordination needs to exist between server threads for client joining and departing

Consider carefully the shared data structures of the server in your answer.

## Answers:

Worth a Piazza post : discuss online...

## Course Evals

```
CSCI 4061    : Intro to Operating Systems
Lecture 001 : Kauffman
```

- This is my second offering of 4061 and while I am aware of some things that went well and some that went poorly, **your feedback is extremely helpful** to making the next offering better
- Fill out evals hand them in
- Need a **volunteer** to deliver them to the CS main office front desk in Keller 4-192