CS2106 Operating Systems I

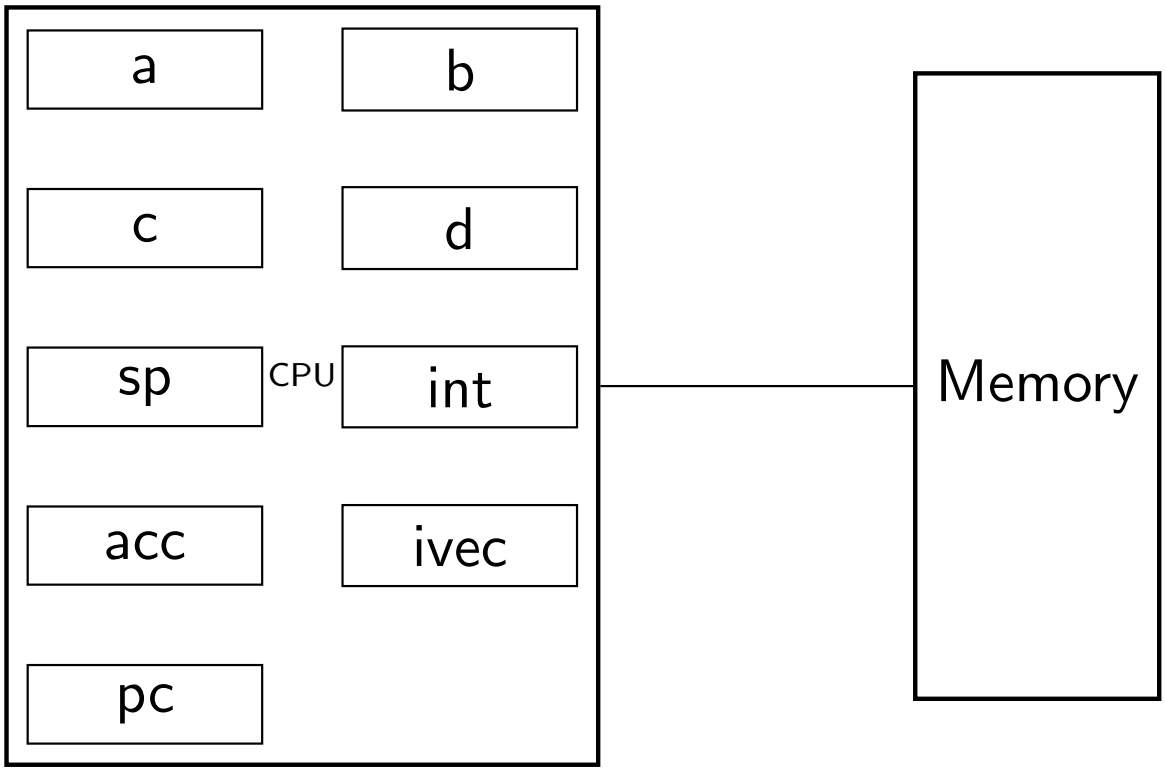
Dr. Chamath Keppitiyagama

University of Colombo School of Computing

Evaluation

- Assignments (1 per week) 40%
- Final Paper 60%

The Machine



Boot Process

- MBR
- Boot sector
- Boot loader
- init process
- Startup scripts

Compiling, Linking and Loading

Process

- Program
- Process
- Thread
- Text, Data, Heap, Stack
- Context Switch
- PCB
- Process states

Process Creation

- `fork()`
- `exec()`
- COW

Interprocess Communication

- Information sharing
- Computation speedup
- Modularity
- Convenience

Interprocess Communication

- Shared Memory
- Message Passing

Producer Consumer Problem

```
#define BUFFER_SIZE 10
typedef struct {
        . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Producer

```
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER SIZE) == out)
                ; /* do nothing */
        buffer[in] = next produced;
        in = (in + 1) % BUFFER SIZE;
}
```

Comsumer

```
while (true) {
        while (in == out)
                ; /* do nothing */
        next consumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;

        /* consume the item in next consumed */
}
```

POSIX - IPC - Shared Memory Server

```
main()
{
        char c;
        int shmid;
        key_t key;
        char *shm, *s;

        key = 5678;
        shmid = shmget(key, SHMSZ, IPC_CREAT | 0666);
        shm = shmat(shmid, NULL, 0);
        s = shm;
        for (c = 'a'; c <= 'z'; c++) *s++ = c;
        while (*shm != '*') sleep(1);
        exit(0);
}
```

Adapted from the example in http://www.cs.cf.ac.uk/Dave/C/

POSIX - IPC - Shared Memory Client

```
main()
{
    int shmid;
    key_t key;
    char *shm, *s;

    key = 5678;
    shmid = shmget(key, SHMSZ, 0666);
    shm = shmat(shmid, NULL, 0);
    for (s = shm; *s != NULL; s++)
        putchar(*s);
    putchar('\n');
    *shm = '*';
    exit(0);
}
```

Adapted from the example in http://www.cs.cf.ac.uk/Dave/C/

Pipe

```
main(int argc, char *argv[])
{   int pipefd[2];
    char buf;
    pipe(pipefd);
    if (fork()==0) {/* Child reads from pipe */
        close(pipefd[1]); /* Close unused write e
        while (read(pipefd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);
        write(STDOUT_FILENO, "\n", 1);
        close(pipefd[0]);
    } else { /* Parent writes argv[1] to pipe */
        close(pipefd[0]); /* Close unused read en
        write(pipefd[1], argv[1], strlen(argv[1])
        close(pipefd[1]); /* Reader will see EOF
        wait(NULL);/* Wait for child */
    }
}
```

Named Pipes - FIFO

```
$ mkfifo abcd
$ ls -l > abcd
$ cat < abcd
```

Socket - Server

```
int sockfd, newsockfd, portno,n;
socklen_t clilen;
char buffer[256];
struct sockaddr_in serv_addr, cli_addr;

sockfd = socket(AF_INET, SOCK_STREAM, 0);
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
```

Socket - Server contd ...

```
bind(sockfd, (struct sockaddr *) &serv_addr,
                              sizeof(serv_addr));
listen(sockfd,5);
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd, (struct sockaddr *)
                              &cli_addr, &clilen);

n = read(newsockfd,buffer,255);
n = write(newsockfd,"HELLO",6);
```

Socket - Client

```
int sockfd, portno, n;
struct sockaddr_in serv_addr;
struct hostent *server;
char buffer[256];

portno = atoi(argv[2]);
sockfd = socket(AF_INET, SOCK_STREAM, 0);

server = gethostbyname(argv[1]);

bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
    (char *)&serv_addr.sin_addr.s_addr,
    server->h_length);
serv_addr.sin_port = htons(portno);
```

Socket - Client contd ...

```
connect(sockfd,(struct sockaddr *) &serv_addr,siz

n = write(sockfd,buffer,strlen(buffer));

bzero(buffer,256);
n = read(sockfd,buffer,255);
```

RPC

- Marshalling
- Big-Endian (MSB first) and Little-Endian (LSB first)
- *exactly once* and *at most once*
- Idempotent operations
- XDR

Synchronization

The Critical-Section Problem

- Mutual exclusion
- Progress
- Bounded waiting

Solutions ???

- Disabling Interrupts
- lock
- Strict Alteration

Bounded-Waiting Mutual Exclusion

```
do{
        waiting[i]=TRUE;
        key=TRUE;
        while(waiting[i] && key)
                key=TestAndSet(&lock);
        waiting[i]=FALSE;
        // Critical Section
        j = (i+i) % n;
        while((j!=i)&&!waiting[j])
                j=(j+1) % n;
        if (j==i)
                lock=FALSE;
        else
                waiting[j]=FALSE;
        // Remainder section
} while(TRUE);
```

Dining Philosophers

```
void philosopher(int i){
 while(TRUE){
            think();
            take_fork(i);
            take_fork((i+1)%N);
            eat();
            put_fork(i);
            put_fork((i+1)%N)
          }
}
```

Dining Philosophers

```
void philosopher(int i){
 while(TRUE){
        think();
        take_forks(i);
        eat();
        put_forks(i);
}}

void take_forks(i){
        down(&mutex);
        state[i]=HUNGRY;
        test(i);
        up(&mutex);
        down(&s[i]);
}
```

Dining Philosophers

```
void put_forks(i){
        down(&mutex);
        state[i]=THINKING;
        test(LEFT);
        test(RIGHT);
        up(&mutex);
}

void test(i){
  if (state[i]==HUNGRY && state[LEFT]!=EATING
       && state[RIGHT]!=EATING)
  {
        state[i]=EATING;
        up(&s[i]);
  }
}
```

Readers Writers - Reader

```
void reader(){
 while(TRUE){
            down(&mutex);
            rc=rc+1;
            if (rc==1)
                  down(&db);
            up(&mutex);
            read_db();
            down(&mutex);
            rc=rc-1;
            if (rc==0)
                  up(&db);
            up(&mutex);
            // other work
          }
}
```

Readers Writers - Writer

```
void writer(){
 while(TRUE){
            //Generate data
            down(&db);
            write_db();
            up(&db);
            // other work
          }
}
```

Producer Consumer - Producer

```
full=0;
empty=N;
mutex=1;

void producer(){
 while(TRUE){
            //Produce and item
            down(&empty);
            down(&mutex);
            // Insert item
            up(&mutex);
            up(&full);
          }
}
```

Producer Consumer - Consumer

```
void consumer(){
 while(TRUE){
            down(&full);
            down(&mutex);
            // Remove an item
            up(&mutex);
            up(&empty);
          }
}
```