BITS Pilani
Pilani Campus

Network Programming

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani Campus

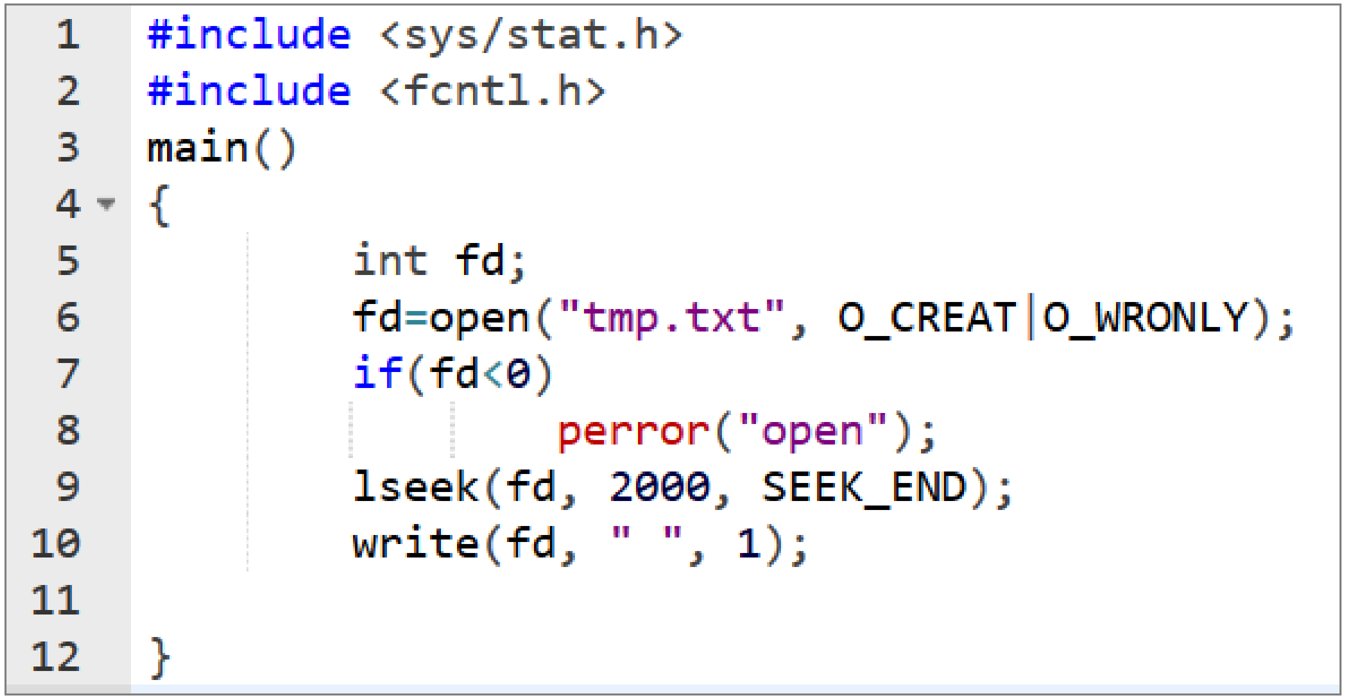
Midterm review

File I/O

- File descriptors
- Sys calls for files
  - open read()
  - write() close()
  - lseek()
- File holes
- fcntl, ioctl
- Open File Table
- dup(), dup2()
- Standard I/O
- *stdio* buffering
  - Buffer modes
  - setvbuf
  - setbuf()
- Mixing fds and file streams
  - fileno()
  - fdopen()

Exercises

- Open a file ‘tmp.txt’ and create a file hole of 2000 bytes?



```c
#include <sys/stat.h>
#include <fcntl.h>
main()
{
        int fd;
        fd=open("tmp.txt", O_CREAT|O_WRONLY);
        if(fd<0)
                perror("open");
        lseek(fd, 2000, SEEK_END);
        write(fd, " ", 1);

}
```

Exercises

- What is the difference between a file descriptor and FILE*?
- Why Standard I/O library provides buffering?

- Kernel gives a fd whenever we open a file. We can open a file stream on top of a fd. File stream has a buffer that pre-reads to reduce the number of system calls.

Exercises

- Implement redirection *‘ls>listoffiles.txt’*

Process Management

- Memory layout
- Env variables
- Cmd line arguments
- setjmp(), longjmp()
- fork()
  - copy-on-write
  - vfork()
- wait(), waitpid()
- zombies,& orphan processes
- Loading a program
  - execve()
  - file descripors
  - Signal handlers

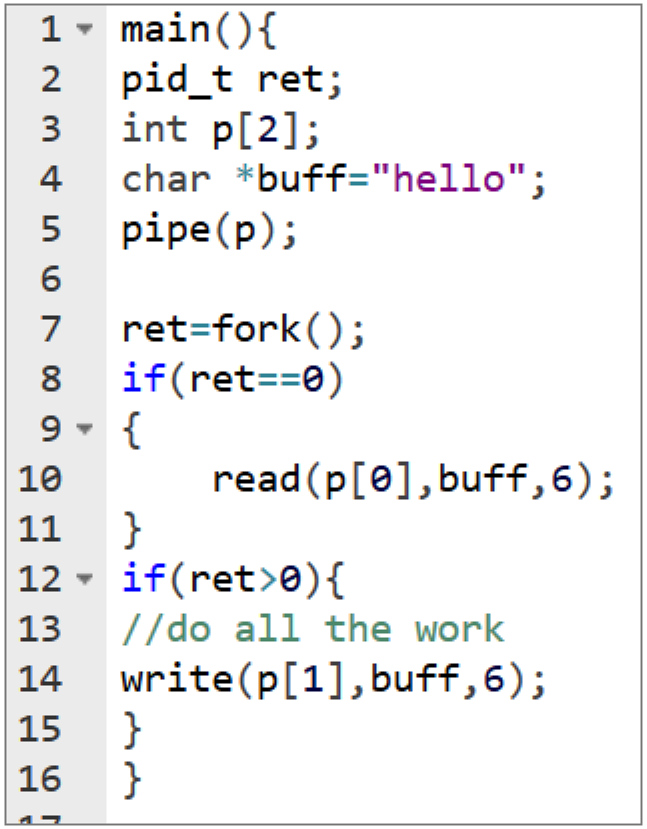
Exercises

- Explain the steps that take place when '*ls -l*' command is executed in UNIX shell?

- 1. create a child process.
- 2. create new process group
- 3. use execv to load the ls program. Pass "ls" and "-l" as parameters.

Exercises

- Write a program that creates a child and the child waits till the parent completes execution?

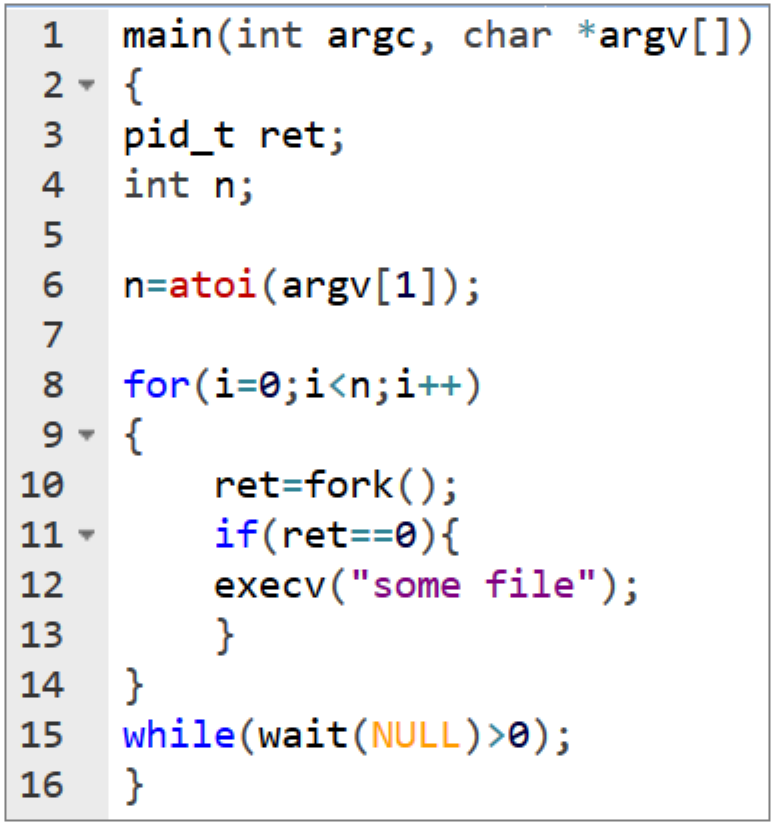


```
main(){
pid_t ret;
int p[2];
char *buff="hello";
pipe(p);

ret=fork();
if(ret==0)
{
	read(p[0],buff,6);
}
if(ret>0){
//do all the work
write(p[1],buff,6);
}
}
```

Exercises

- Write a program that creates *n* children and waits until every child exits. *n* is given as command-line argument to your program.



```
main(int argc, char *argv[])
{
pid_t ret;
int n;

n=atoi(argv[1]);

for(i=0;i<n;i++)
{
	ret=fork();
	if(ret==0){
	execv("some file");
	}
}
while(wait(NULL)>0);
}
```

Exercises

- What is the purpose of the WNOHANG option of the waitpid() system call?

- waitpid will return immediately without waiting for the child.

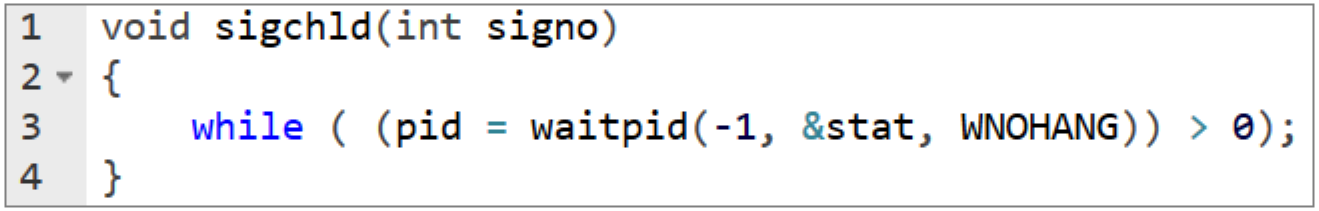
Exercises

- Explain what happens if a parent process does not 'wait' for the termination of a child process?
- Child process will become a zombie process.

Exercises

- What is a zombie process? How does a server designed to provide concurrency using a multi-process model would face problems with zombie processes? Provide a server program that avoids zombie processes

- In SIGCHLD Signal handler:



```
void sigchld(int signo)
{
    while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0);
}
```

Exercise

- When is it possible for a child to modify parent’s memory? [hint: vfork()]

Exercises

- Create a tree of processes

- P1->p2
- P1->p3
- P2->p4
- P3->p5,p6
- P6->p7

Signals

- What is a Signal?
- Signal disposition
- Signal generation
- Signal delivery
- Signal pending
- Signal mask
- Sigaction
- Waiting for a signal
  - Sigpending()
  - Pause()

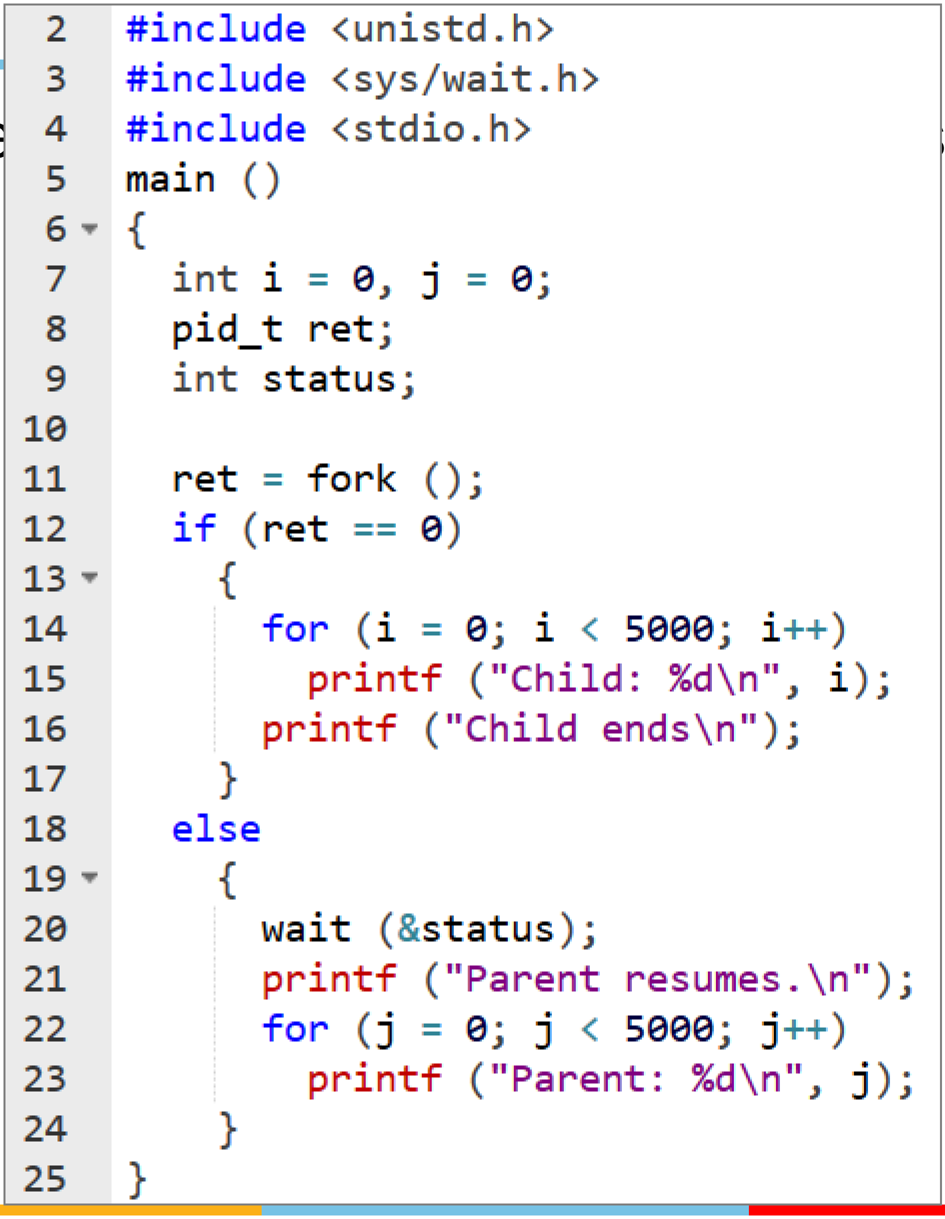
Exercises

- Give example of signals generated from software events, hardware exceptions, user generated?

- What are the administrative signals?

Exercises

- Synchronize betwe ?

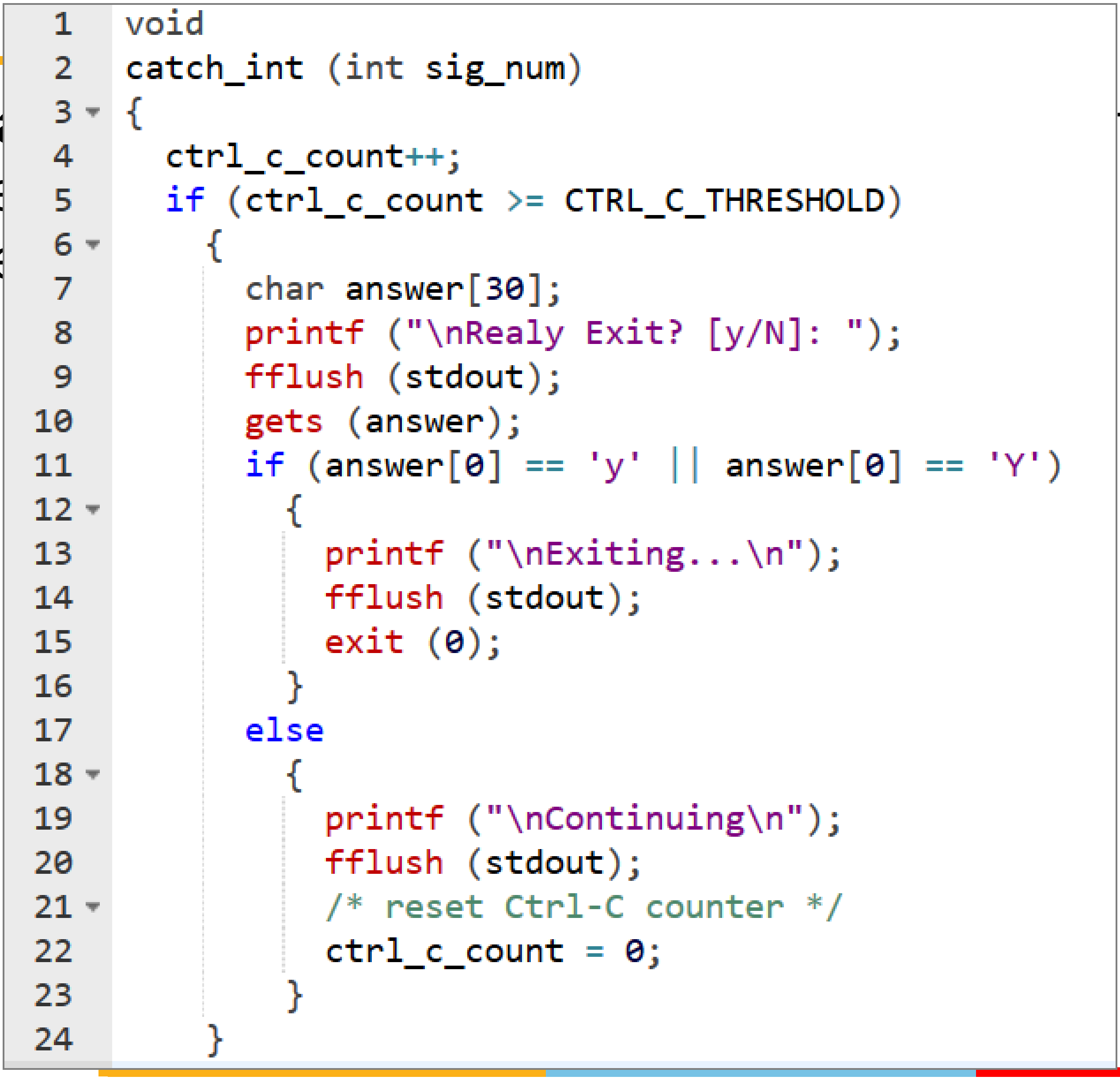


```
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
main ()
{
  int i = 0, j = 0;
  pid_t ret;
  int status;

  ret = fork ();
  if (ret == 0)
    {
      for (i = 0; i < 5000; i++)
        printf ("Child: %d\n", i);
      printf ("Child ends\n");
    }
  else
    {
      wait (&status);
      printf ("Parent resumes.\n");
      for (j = 0; j < 5000; j++)
        printf ("Parent: %d\n", j);
    }
}
```

Exercises

- Write a trl-C is presse reache



```
void
catch_int (int sig_num)
{
  ctrl_c_count++;
  if (ctrl_c_count >= CTRL_C_THRESHOLD)
    {
      char answer[30];
      printf ("\nRealy Exit? [y/N]: ");
      fflush (stdout);
      gets (answer);
      if (answer[0] == 'y' || answer[0] == 'Y')
        {
          printf ("\nExiting...\n");
          fflush (stdout);
          exit (0);
        }
      else
        {
          printf ("\nContinuing\n");
          fflush (stdout);
          /* reset Ctrl-C counter */
          ctrl_c_count = 0;
        }
    }
```

Exercises

- Block SIGINT signal and atomically unblock and wait for the delivery of the signal?

File Permissions & Process Groups

- Uid, gid
- User id
  - Real
  - Effective
  - Saved set user id
- Set-user-id
- Sticky bit
- umask()
- Process groups
- Sessions

Exercises

- How does Linux maintain time?
- What is a process group?
- What is a session?
- What are the signals involved with job control?

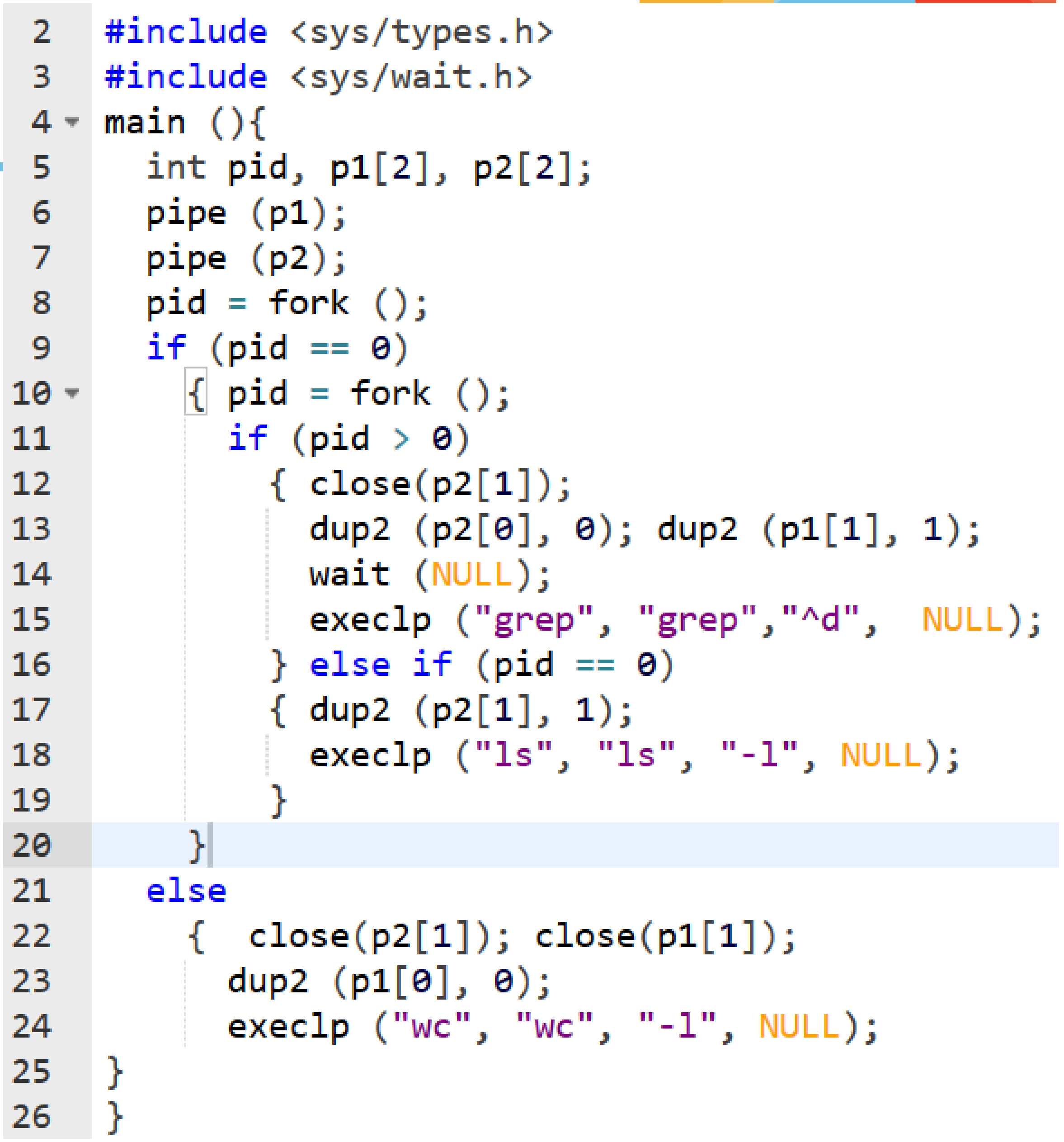
IPC

- Pipes & FIFO
- System V IPC
  - Msg Qs
  - Semaphores
  - Shared memory

Exercise

Write a program to implement

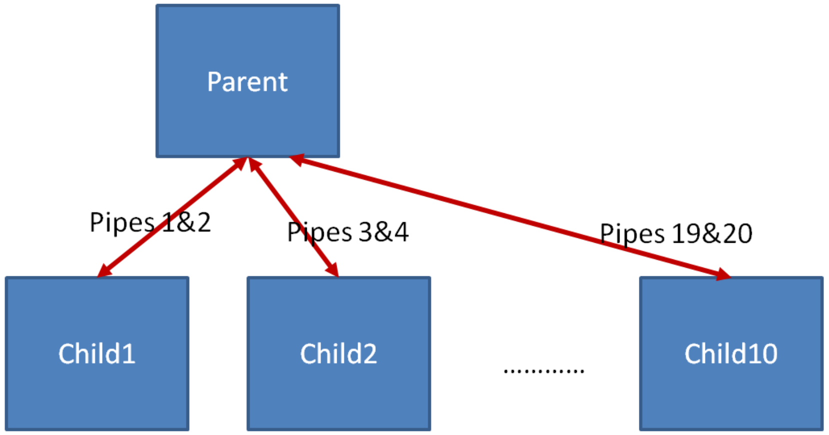
ls –l|grep ^d|wc -l

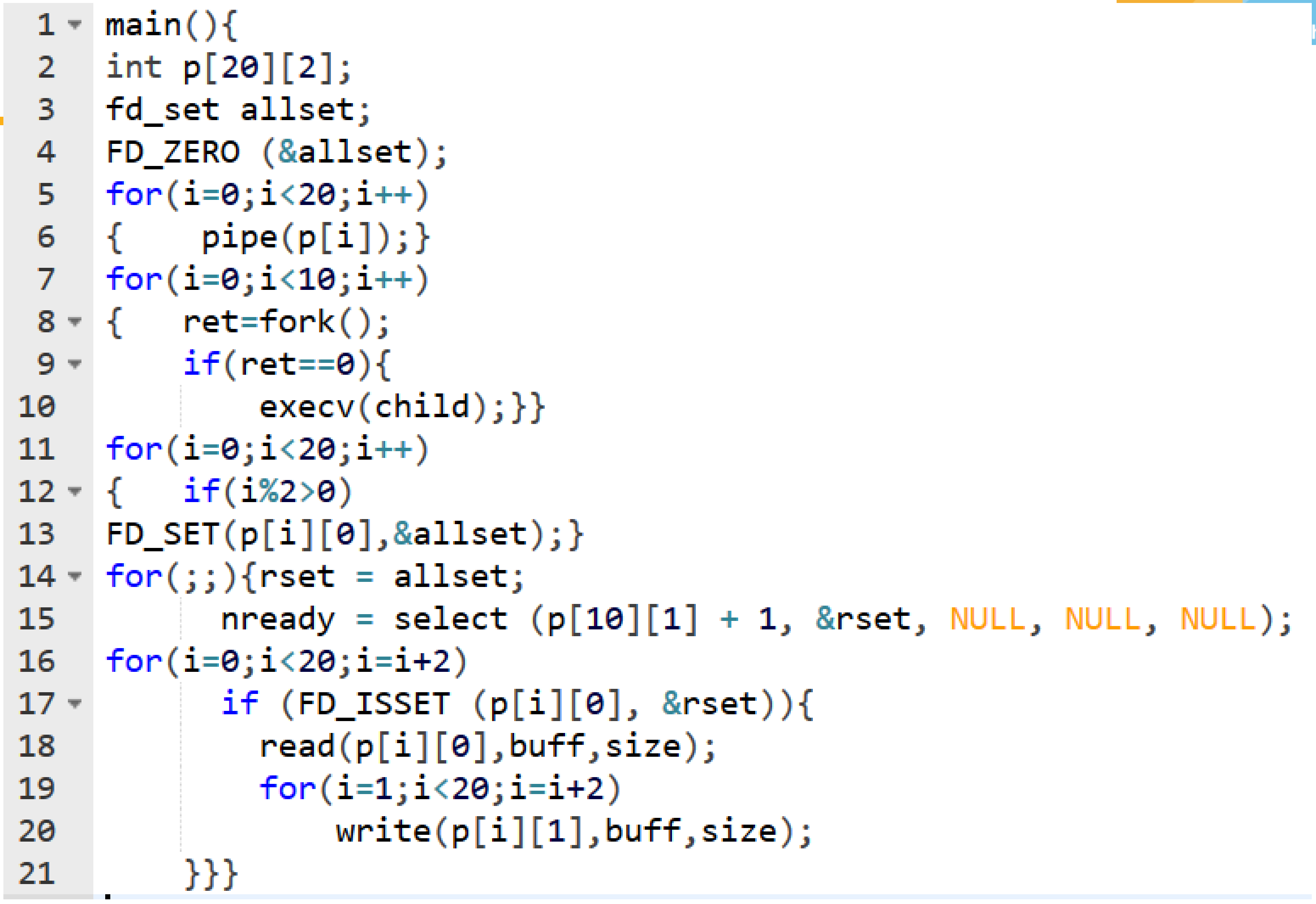


```c
#include <sys/types.h>
#include <sys/wait.h>
main (){
  int pid, p1[2], p2[2];
  pipe (p1);
  pipe (p2);
  pid = fork ();
  if (pid == 0)
    { pid = fork ();
      if (pid > 0)
        { close(p2[1]);
          dup2 (p2[0], 0); dup2 (p1[1], 1);
          wait (NULL);
          execlp ("grep", "grep","^d",  NULL);
        } else if (pid == 0)
        { dup2 (p2[1], 1);
          execlp ("ls", "ls", "-l", NULL);
        }
    }
  else
    {  close(p2[1]); close(p1[1]);
      dup2 (p1[0], 0);
      execlp ("wc", "wc", "-l", NULL);
}
}
```

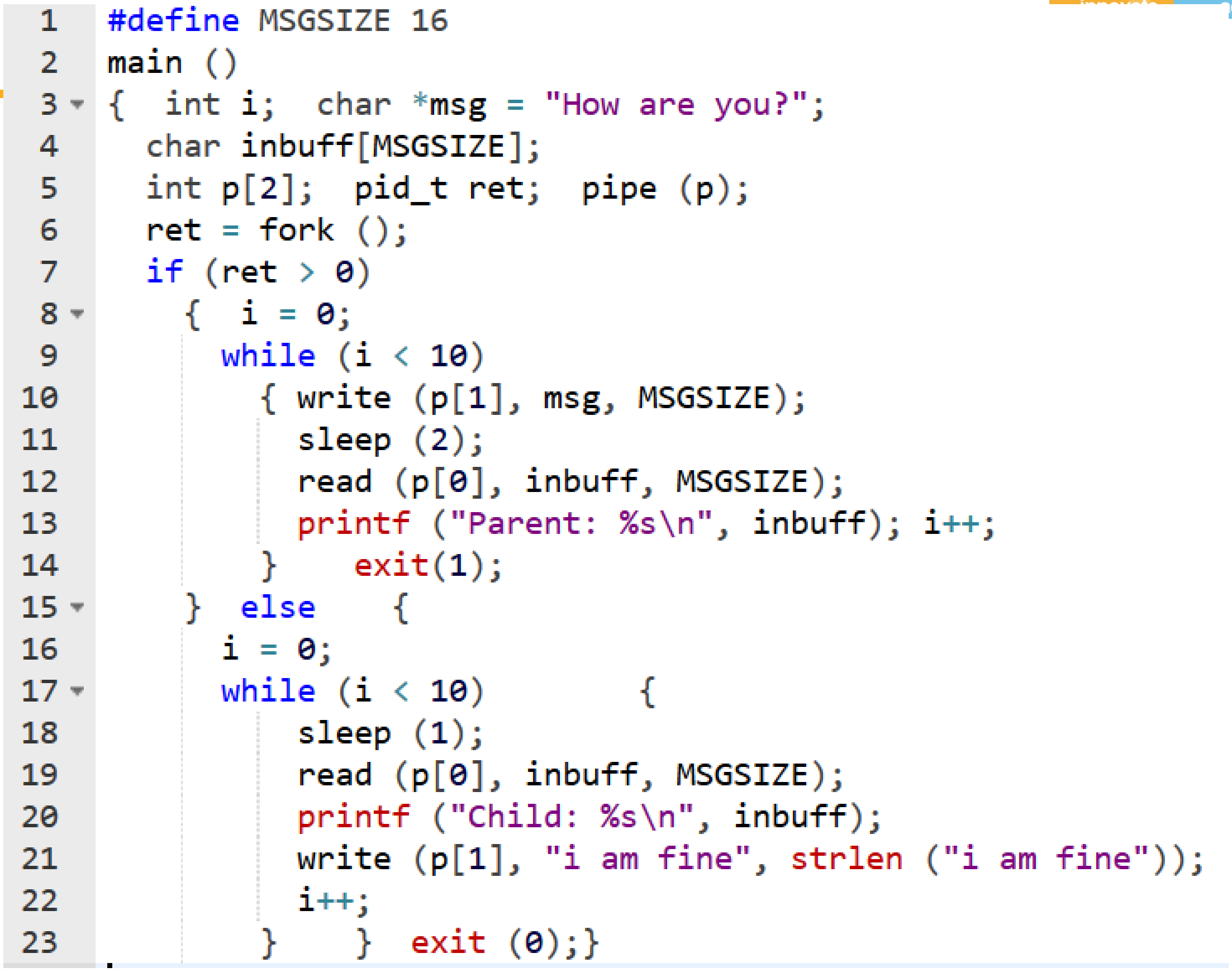
Exercise

- Let us consider designing a system in which a parent has 10 children. Whenever a child
- i) Parent sets up 20 pipes.
- ii) Parent creates 10 children. For child1 pipe1 and pipe 2 is used for duplex communication. For child2, pipe3 and pipe4 are used. In the same way other pipes are used.
- iii) Child communicates with parent through odd numbered pipe and parent communicates with child using even numbered pipe.
- iv) If any child sends a message to parent, parent sends this message to all other children.





```
main(){
int p[20][2];
fd_set allset;
FD_ZERO (&allset);
for(i=0;i<20;i++)
{    pipe(p[i]);}
for(i=0;i<10;i++)
{   ret=fork();
    if(ret==0){
        execv(child);}}
for(i=0;i<20;i++)
{   if(i%2>0)
FD_SET(p[i][0],&allset);}
for(;;){rset = allset;
      nready = select (p[10][1] + 1, &rset, NULL, NULL, NULL);
for(i=0;i<20;i=i+2)
      if (FD_ISSET (p[i][0], &rset)){
        read(p[i][0],buff,size);
        for(i=1;i<20;i=i+2)
            write(p[i][1],buff,size);
    }}}
```



```
#define MSGSIZE 16
main ()
{  int i;  char *msg = "How are you?";
   char inbuff[MSGSIZE];
   int p[2];  pid_t ret;  pipe (p);
   ret = fork ();
   if (ret > 0)
     {  i = 0;
        while (i < 10)
          { write (p[1], msg, MSGSIZE);
            sleep (2);
            read (p[0], inbuff, MSGSIZE);
            printf ("Parent: %s\n", inbuff); i++;
          }    exit(1);
     }  else    {
        i = 0;
        while (i < 10)       {
            sleep (1);
            read (p[0], inbuff, MSGSIZE);
            printf ("Child: %s\n", inbuff);
            write (p[1], "i am fine", strlen ("i am fine"));
            i++;
          }    }  exit (0);}
```

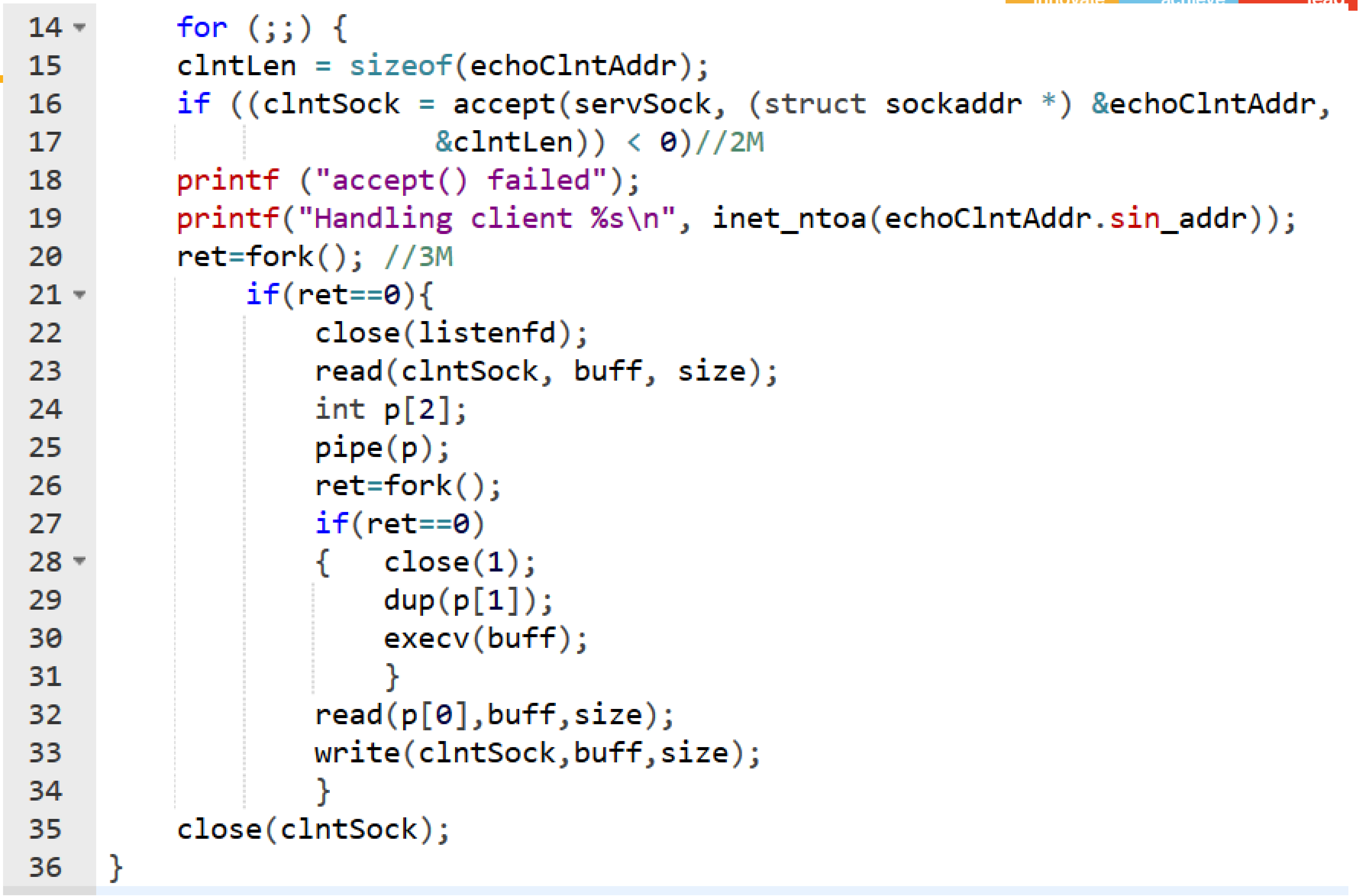
TCP Client Server

- TCP connection
  - Establishment
  - Termination
  - Connection states
- Client
- server

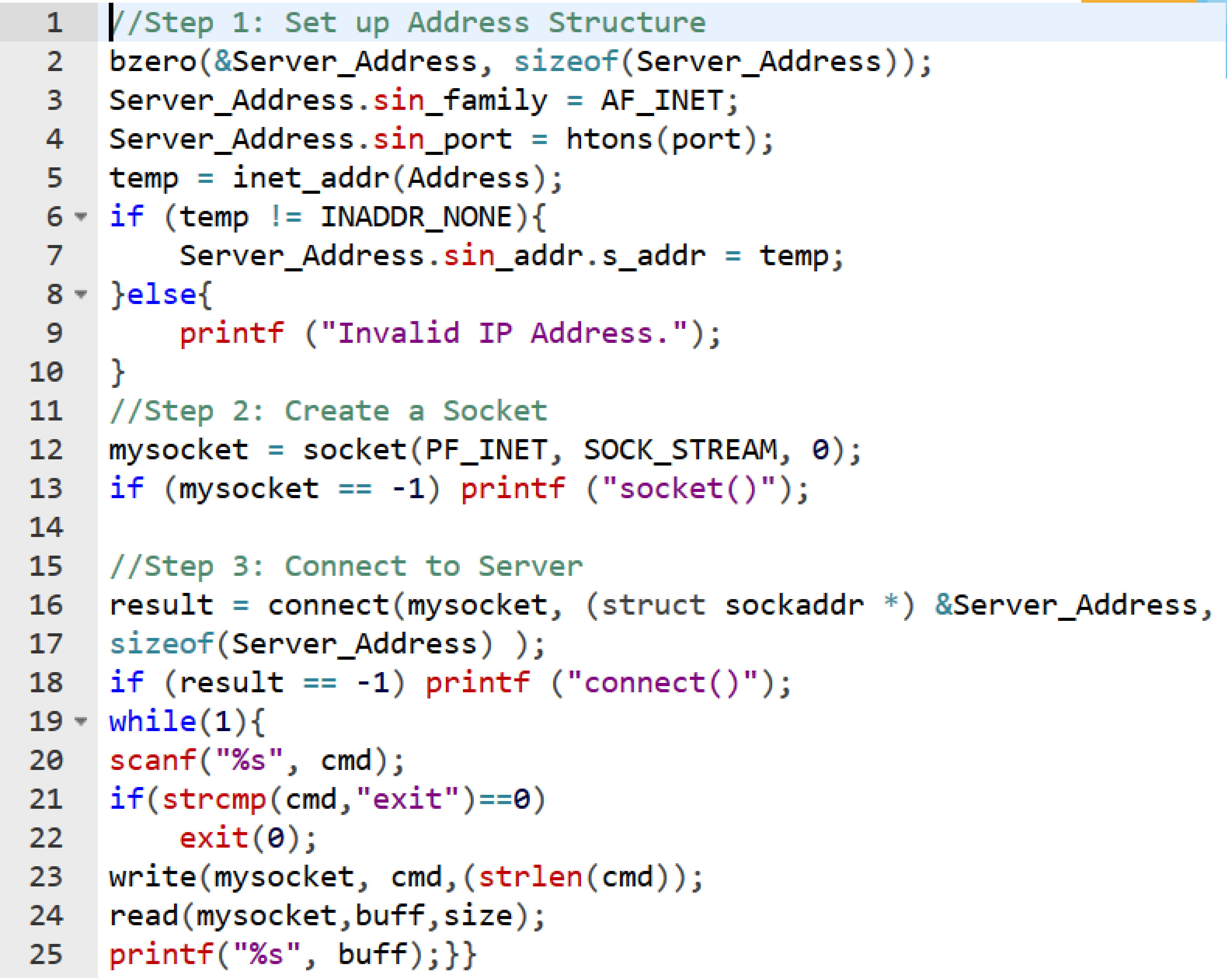
Exercise

- Write a TCP client and server that fulfills the following requirements.
- Server.c:
    - server should take port number on command-line and listen on that port.
    - server should create a child to handle a new client.
    - When a client sends a command such as 'ps', server should execute the command and send the output to the client.
    - it should take care of zombies.
- Client.c:
    - client takes ip address and port number of the server on command-line.
    - client sets up a connection to the server.
    - client takes a command from the user and sends it to the server.
    - client waits for the reply and prints the reply on the standard output.

```
if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
printf("socket() failed");
memset(&echoServAddr, 0, sizeof(echoServAddr));
echoServAddr.sin_family = AF_INET;
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
echoServAddr.sin_port = htons(echoServPort);
if (bind
(servSock, (struct sockaddr *) &echoServAddr,
 sizeof(echoServAddr)) < 0)
printf("bind() failed");
if (listen(servSock, MAXPENDING) < 0)
printf("listen() failed");
```



```
    for (;;) {
    clntLen = sizeof(echoClntAddr);
    if ((clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,
                   &clntLen)) < 0)//2M
    printf ("accept() failed");
    printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));
    ret=fork(); //3M
        if(ret==0){
            close(listenfd);
            read(clntSock, buff, size);
            int p[2];
            pipe(p);
            ret=fork();
            if(ret==0)
            {   close(1);
                dup(p[1]);
                execv(buff);
                }
            read(p[0],buff,size);
            write(clntSock,buff,size);
            }
    close(clntSock);
}
```



```c
//Step 1: Set up Address Structure
bzero(&Server_Address, sizeof(Server_Address));
Server_Address.sin_family = AF_INET;
Server_Address.sin_port = htons(port);
temp = inet_addr(Address);
if (temp != INADDR_NONE){
    Server_Address.sin_addr.s_addr = temp;
}else{
    printf ("Invalid IP Address.");
}
//Step 2: Create a Socket
mysocket = socket(PF_INET, SOCK_STREAM, 0);
if (mysocket == -1) printf ("socket()");

//Step 3: Connect to Server
result = connect(mysocket, (struct sockaddr *) &Server_Address,
sizeof(Server_Address) );
if (result == -1) printf ("connect()");
while(1){
scanf("%s", cmd);
if(strcmp(cmd,"exit")==0)
    exit(0);
write(mysocket, cmd,(strlen(cmd));
read(mysocket,buff,size);
printf("%s", buff);}}
```

I/O Models

- Blocking Model
- Non-blocking Model
  - Non-blocking read/write or select() with non-blocking I/O
  - Web client with non-blocking connect()
- I/O Multiplexing
  - Readability/writability
  - Select() with blocking i/o
  - Server with select()
- Signal Driven I/O
- Asynchronous I/O

Unix domain sockets

- Advantages
- Stream and datagram sockets
- Socket pair
- Passing fds
- Passing credentials

Threads

- Thread management
- Mutexes
- Condition variables

- Concurrent servers using I/O multiplexing based concurrency have the advantage of shared memory and the draw-back of instability and inability to span across multiple cores or processers. But to have stability and scalability in the system, it is planned that the server will use combination of forking with select() as the model for concurrency. No. of clients per process are defined to be *CPP*. Server pre-forks *N* processes and each process can accept up to *CPP* number of clients. Assume a simple chat protocol that client sends three messages JOIN <name>, LEAV and CHAT <targetname> <msg>. In join message, client sends its nickname and in chat it sends the nickname of the person to whom it needs to be delivered and the contents. Define a model for the server with the help of a diagram highlighting the data structures and IPC. Write a program for the server (parent & child) that achieves concurrency for this protocol.

Acknowledgements

Q&A

BITS Pilani
Pilani Campus

Thank You