Interprocess Communication

- Reading: OS Concepts pp.107-124, pp.201-206
- A way of allow processes
  - To commincate
  - To synchronise
- Types:
  - Shared Memory
  - Semaphores
  - Pipes & FIFOs
  - Signals
  - Message Passing…
    - Sockets
    - Remote Procedure Calls

IPC Design Issues

- Naming. Can be
  - *Direct* where processes are named
    - _____
    - Sender always specifies receiver but comms can be
      - _____
      - _____
  - Indirect where mailbox/port is used
    - _____
    - _____
- Synchronisation for Send and Receive
  - Blocking Send:_____
  - Non-blocking Send:_____
  - Blocking Receive:_____
  - Non-blocking Receive:_____

IPC Design Issues

- Buffering capacity
  - _____
  - _____
  - _____
- Data Format
  - Structure: _____
  - Length: _____
- Queueing mechanism
  - _____
  - _____

Shared Memory

- Memory is automatically shared between threads in the same process
- For multiple processes we explicitly set up shared memory areas
  - shmget() _____
  - shmat() _____
  - shmctl() _____
- Naming: _____
- Synchronisation: _____
- Buffering capacity: _____
- Data Format: _____
- Queueing mechanism: _____
- Most appropriate for: _____

Shared Memory Example

Resources/Code/SharedMemoryExample.C

```
main (){
  int SharedMemoryID;
  char* SharedMemoryAddress;
  if ((SharedMemoryID = shmget((key_t) IPC_PRIVATE,
        sizeof(char), 0666 | IPC_CREAT)) == -1)
    syserr("shmget");
  if ((SharedMemoryAddress = (char *)
      shmat(SharedMemoryID, 0, 0)) == (void *)(-1))
    syserr("shmat");
  *SharedMemoryAddress = 'a';
  if (shmctl(SharedMemoryID,IPC_RMID,NULL) == -1)
    syserr("shmctl");
}
```


Semaphores

- Basic Operations:
  - Wait: _____
    _____
  - Signal: _____
- _____
- _____
- UNIX system calls:
  - semget() _____
  - semctl() _____
  - semop() _____

Semaphores

- Naming is _____
- Synchronisation is _____
- Buffering capacity is _____
- Data Format is _____
- Queueing mechanism is _____
- Most appropriate for _____

Semaphores Example

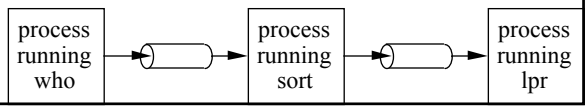
Resources/Code/SemaphoreExample.C

```
main (){
  int semaphore_id;
  if ((semaphore_id = semget((key_t) IPC_PRIVATE,
            1, 0666 | IPC_CREAT)) == -1)
    syserr("semaphore creation");
  union semun {
    int val; struct semid_ds *buf; u_short *array;
    } arg;
  arg.val = 1;
  if (semctl(semaphore_id,0,SETVAL,arg) == -1)
    syserr("semaphore set initial value");
  struct sembuf sb;
  sb.sem_num = 0;  sb.sem_op = 1;  sb.sem_flg = 0;
  if (semop(semaphore_id, &sb, 1) == -1)
    syserr("signal");
  sb.sem_op = -1;   // Operation.
  if (semop(semaphore_id, &sb, 1) == -1)
    syserr("wait");
  semctl(semaphore_id,0,IPC_RMID);
}
```

Pipes & FIFOs

- Unidirectional communication link
- Accessed through 2 file descriptors
- _____
- _____
- UNIX system calls:
  - pipe() _____
  - mkfifo() _____
  - open() _____
  - dup() _____

who | sort | lpr



Pipes & FIFOs

- Naming is _____
- Synchronisation is _____
- Buffering capacity is _____
- Data Format is _____
- Queueing mechanism is _____
- Most appropriate for _____

Pipes Example

Resources/Code/PipeExample.C

```
main (){
  int pfd[2];
  if (pipe(pfd) == -1)
    syserr("pipe");
  int new_pid = fork();
  if (new_pid == 0) {
    if ((close(pfd[0]) == -1) ||
      (close(STDOUT_FILENO) == -1) ||
      (dup(pfd[1]) == -1) || (close(pfd[1]) == -1))
      syserr("closing … and duping output");
    execlp("/usr/local/bin/who","who",NULL);
  }
  else {
    if ((close(pfd[1]) == -1) ||
      (close(STDIN_FILENO) == -1) ||
      (dup(pfd[0]) == -1) || (close(pfd[0]) == -1))
      syserr("closing … and duping input");
    execlp("/bin/sort","sort",NULL);
  }
}
```


Signals

- Akin to an interrupt
- Send a signal using a system call
- Interrupt handling to deal with the signal
- UNIX system calls:
  - kill() _____
  - signal() _____
  - pause() _____
  - alarm() _____

Signals

- Naming is _____
- Synchronisation is _____
- Buffering capacity is _____
- Data Format is _____
- Queueing mechanism is _____
- Most appropriate for _____

Signals Example

Resources/Code/SignalExample.C

```
void handle_signal(int signal_no){
  switch (signal_no) {
    case SIGALRM:
      break;
    case SIGINT:
      cout << " Are you sure you want to quit? ";
      alarm(5);
      while (((no_read = read(STDIN_FILENO, buffer,
       BUFFER_SIZE)) == -1) && (errno == EINTR)) {
        cout << " Are you sure you want to quit? ";
        alarm(5);
      }
      alarm(0);
      if ((buffer[0] == 'y') || (buffer[0] == 'Y'))
        exit(1);
      break;
    ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... .....
 if ((signal(SIGINT, handle_signal) == SIG_ERR) ||
     (signal(SIGALRM, handle_signal) == SIG_ERR))
  syserr("signal");
```

Basic Message Passing

- Basic operations: *Send* and *Receive*
- Message type
  - _____
  - _____
    _____
- If the message too big for the provided buffer then it is truncated to the buffer length
- UNIX system calls:
  - msgget() _____
  - msgsnd() _____
  - msgrcv() _____
  - msgctl() _____

Basic Message Passing

- Naming is _____
- Synchronisation is _____
- Buffering capacity is _____
- Data Format is _____
- Queueing mechanism is _____
- Most appropriate for _____

Message Passing Example

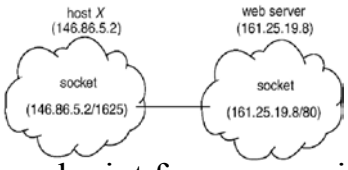

```
main (){
    int msqid;
    if ((msqid = msgget(IPC_PRIVATE,
                        IPC_CREAT | 0666))== -1)
        syserr("Message Queue Creation failed");
    struct MessageFormat {
        long message_type;
        char message_text[MAX_SIZE];
    } message;

    int new_pid;
    if ((new_pid = fork()) == 0) {
        message.message_type = 1;
        strcpy(message.message_text,"The message");
        if ((msgsnd(msqid,&message,sizeof(message),
                                          0)) == -1)
            syserr("Message send failed");
        exit(1);
    }
```


Message Passing Example (continued)


```
else if (new_pid > 0) {
        if ((msgrcv(msqid,&message,sizeof(message),
                                       1,0)) == -1)
            syserr("Message receipt failed");
        cout << "Message received was " <<
                    message.message_text << endl;
        struct msqid_ds buffer;
        if (msgctl(msqid,IPC_RMID,&buffer) == -1)
            syserr("Message Q Deletion failed");
    }
}
```

Sockets



- A socket is an endpoint for communication across a network
- _____
- _____
- Naming: _____
- Synchronisation: _____
- Buffering capacity: _____
- Data Format: _____
- Queueing mechanism: _____
- Most appropriate for: _____

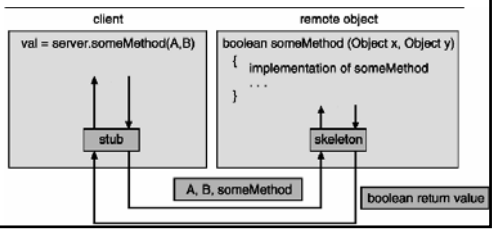
Sockets Example


```
ServerSocket sock = new ServerSocket(5155);
while (true) {
  Socket client = sock.accept();
  PrintWriter pout = new PrintWriter(
           client.getOutputStream(), true);
  pout.println("Welcome");
  pout.close;
  client.close;
}
```

```
Socket sock = new Socket("127.0.0.1",5155);
InputStream in = sock.getInputStream();
BufferedReader bin = new BufferedReader(
           new InputStreamReader(in));
String line;
while ((line = bin.readLine()) != NULL)
  System.out.println(line);
sock.close;
```

Remote Procedure Calls

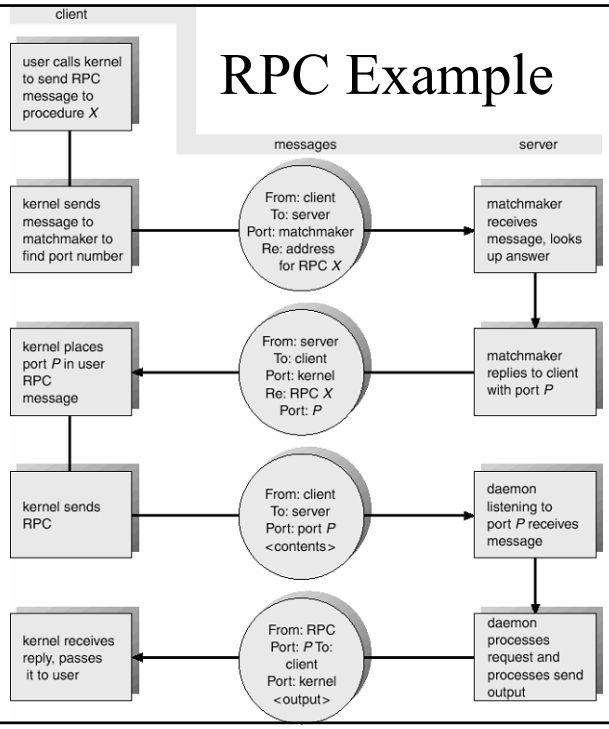
- A form of message passing for making procedure calls on remote hosts
- _____
- _____
- _____
  _____
- _____
- _____
  _____
  _____
  _____



Remote Procedure Calls

- Naming is _____
- Synchronisation is _____
- Buffering capacity is _____
- Data Format is _____
- Queueing mechanism is _____
- Most appropriate for _____
  _____

RPC Example


client
messages
server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client To: server Port: matchmaker Re: address for RPC X
matchmaker receives message, looks up answer
kernel places port P in user RPC message
From: server To: client Port: kernel Re: RPC X Port: P
matchmaker replies to client with port P
kernel sends RPC
From: client To: server Port: port P <contents>
daemon listening to port P receives message
kernel receives reply, passes it to user
From: RPC Port: P To: client Port: kernel <output>
daemon processes request and processes send output