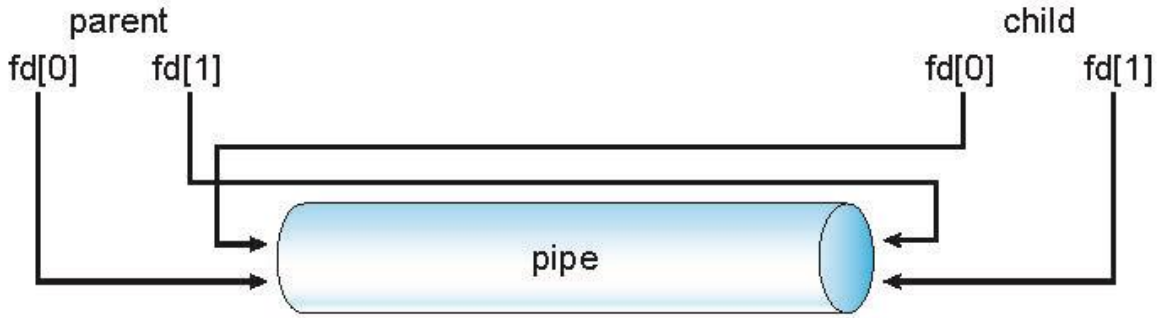
Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these **anonymous pipes**
- See Unix and Windows code samples in textbook

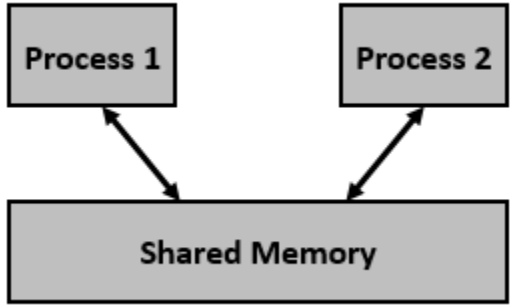
```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define BUFFER SIZE 25
#define READ END 0
#define WRITE END 1
int main(void)
{
char write msg[BUFFER SIZE] = "Greetings";
char read msg[BUFFER SIZE];
int fd[2];
pid t pid;
/* create the pipe */
if (pipe(fd) == -1) {
fprintf(stderr,"Pipe failed");
return 1;
}
```

```
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}
if (pid > 0) { /* parent process */
/* close the unused end of the pipe */
close(fd[READ END]);
/* write to the pipe */
write(fd[WRITE END], write msg, strlen(write msg)+1);
/* close the write end of the pipe */
close(fd[WRITE END]);
}
else { /* child process */
/* close the unused end of the pipe */
close(fd[WRITE END]);
/* read from the pipe */
read(fd[READ END], read msg, BUFFER SIZE);
printf("read %s",read msg);
/* close the write end of the pipe */
close(fd[READ END]);
}
return 0;
}
```

Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

IPC through shared memory



- Create the shared memory segment or use an already created shared memory segment (shmget())
- Attach the process to the already created shared memory segment (shmat())
- Detach the process from the already attached shared memory segment (shmdt())
- Control operations on the shared memory segment (shmctl())

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    cout<<"Write Data : ";
    gets(str);
    printf("Data written in memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    return 0;
}
```

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Data read from memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);
    return 0;
}
```

End of Chapter 3