INTERPROCESS COMMUNICATION

1) Ordinary Pipes:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#define READ_END 0
#define WRITE_END 1


// writing in parent, reading in child


int main() {
   char write_msg[20];
   char read_msg[20];
   int fd[2];
   pid_t pidP, pidF;

   // system call = pipe
   pidP = pipe(fd);
   if (pidP == -1 ) {
        perror("pipe");
        exit(EXIT_FAILURE);
   }

   // system call = fork
   pidF = fork();
   if (pidF == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
   } else if (pidF == 0) {
        // child
        close(fd[WRITE_END]);
        read(fd[READ_END], read_msg, 20);
        printf("Data Read: %s\n", read_msg);
        close(fd[READ_END]);
    } else {
```

```
        // parent
        close(fd[READ_END]);
        printf("Enter Data: ");
        scanf("%s", write_msg);
        write(fd[WRITE_END], write_msg, strlen(write_msg) +
1);
        close(fd[WRITE_END]);
    }
}
```

Takeaways:

- **Uses pipe system call that returns pid_t and takes int fd[2] as parameter.**
- **Any process from parent and child can contain the write or the read functionality.**
- **If write and read enclosed in while(1) loops, it's preferable to sleep(1) after every write so that the read process has time to read.**
- **When writing, close the read end then write to the write end then close the write end.**
- **When reading, close the write end then read from the read end then close the read end.**

2) Named Pipes:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO_FILE "/tmp/myfifo"

int main() {
   int fd;
   char buffer[20];
   ssize_t numBytes;
   pid_t pid;
```

```
    mkfifo(FIFO_FILE, 0666);
    pid = fork();
    if (pid == -1) {
          perror("fork");
          exit(EXIT_FAILURE);
    } else if (pid == 0) {
          // child (read)
          fd = open(FIFO_FILE, O_RDONLY);
          if (fd == -1) {
                perror("open");
                exit(EXIT_FAILURE);
          }
          while(1) {
                numBytes = read(fd, buffer, sizeof(buffer));
                if (numBytes == -1) {
                      perror("read");
                      exit(EXIT_FAILURE);
                }
                printf("Message Read: %s\n", buffer);
          }
          close(fd);
    } else {
          // parent (write)
          fd = open(FIFO_FILE, O_RDWR);
          if (fd == -1) {
                perror("open");
                exit(EXIT_FAILURE);
          }
          while(1) {
                printf("Enter Message To Write: ");
                fgets(buffer, sizeof(buffer), stdin);
                numBytes=write(fd, buffer, strlen(buffer)+1);
                if (numBytes == -1) {
                      perror("write");
                      exit(EXIT_FAILURE);
                }
                sleep(1);
          }
          close(fd);
    }
    unlink(FIFO_FILE);
}
```

Takeaways:

- **Uses mkfifo system call that returns an integer indicating success or failure and takes a pathname and permissions/mode as parameters.**
- **Uses write system call in producer (parent) process.**
- **Uses read system call in consumer (child) process.**

3) Shared Memory:

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

int main() {
   const int SIZE = 1000;
   const char *name = "Shared Name";
   char message[SIZE];
   int fd;
   char *ptr;
   fd = shm_open(name, O_CREAT | O_RDWR, 0666);
   ftruncate(fd, SIZE);
   ptr = (char*) mmap(NULL, SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
   int *pro = (int*)ptr;
   int *con = pro + 1;
   *con = 0;
   *pro = 1;
   close(fd);
   pid_t pid;
   pid = fork();
   if (pid == -1) {
         perror("fork");
         exit(EXIT_FAILURE);
   } else if (pid == 0) {
         // child (consumer)
```

```
        while(1) {
            if(*con == 1) {
                printf("Message Read: %s", (char*)(ptr +
                2*sizeof(int)));
                *con = 0;
                *pro = 1;
                sleep(2);
            }
        }
    } else {
        // parent (producer)
        while(1) {
            if(*pro == 1) {
                printf("Enter message to write: ");
                fgets(message, sizeof(message), stdin);
                sprintf(ptr+2*sizeof(int),"%s", message);
                *con = 1;
                *pro = 0;
                sleep(2);
            }
        }
    }
    shm_unlink(name);
}
```

Takeaways:

- **Implementing shared memory with multiple processes is prone to race condition. Integer flags (stored in the shared memory itself after assigning ptr to mmap) help resolve this.**
- **Sleep is used both in consumer and producer.**
- **Producer uses sprintf to write to shared memory.**
- **Consumer uses the ptr in printf.**
- **If making separate files, don't unlink in producer.**