name	ماهر سليم اخضير
id	1130258

85

Birzeit University - Faculty of Engineering Technology
Electrical & Computer Engineering Department – ENCS531
Real-Time Applications and Embedded Systems - 2st semester - 2016/17

Midterm Exam – APR 9- 2017- duration (80 mins)

Real-Time Applications and Embedded Systems

Instructor: Dr. Ahmad Afaneh

Q1. Select the most correct answer. Write down your final answers in the following table. (18 points) 2 each

1	2	3	4	5	6	7	8	9	
a	b	b	a	a	b	b	b	b	

18

1) Condition variables can cause the release of the mutex inside the critical sectiona.
   a. TRUE b. FALSE

2) Marshalling in RPC is performed by the client stub only
   a. TRUE b. FALSE

3) The Win32 shared memory can be used between different processes on the same network
   a. TRUE b. FALSE

4) Mailslots in Win32 can receive messages from processes on different machine in the same network
   a. True b. False

5) A TCP Server can handle more than one client socket without using threads
   a. True fork( ) b. False

6) In a real time system the correctness of the system depends only on the timing
   a. True b. False output also

7) A pthread can wait for the completion of a nondetached thread by using pthread_wait ()
   a. TRUE b. FALSE

8) The maximum number of message types supported by a message queue is 1024
   a. True b. False 2^{32}

9) When a signal is masked (blocked) it will never be delivered
   a. True b. False

name	
id	1130258

7

Q2. (12 points)

a. What is the main advantage and main disadvantage to using shared memory for IPC?

advantages : its fast since its pointer to address

disadvantages : we have to do everything by hand like Mutex, synchronizing security ...

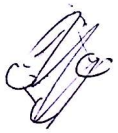
4

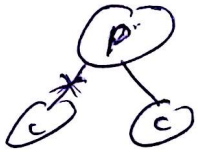
b. When a parent process creates a pipe then calls fork twice, what is the status of the pipe in terms of connections?

the pipe will be connected to ~~both childs~~ one of the childs

~~ie~~ when parent writes data to pipe, one of the childs will take it not both. also when both writes it will take from one of them ⇒ it will be connect to one of them.

(its randomly to what child it will be connected)





2

c. What is the main advantage of using a message queue instead of named pipe in IPC?

in Message queue we can add a "Type" to the message ⇒ when a process requests a Message with type=x ⇒ it will get the first Message in the queue with type x . In contrast in named Pipes there is no pipe ⇒ we can't garntee which process will take the message from FIFO pipe

3

Q3) (70 points) multipart downloader, The idea is to have the file content distributed amongst clients, the system consists of clients (peers) and a server,

1. when the client need to download a file it will connect to the server and send file name the server will respond with text file that contains the information about each part (assume that there is a ready Function GetFileInfo(char*) that does that and save the file in path INFOFILE)
2. The file format is as follows

N // number of file parts
Id size IP port // for each file part
N Lines

3. The client then creates a connection for each file part send the id and and download the content (each connection in new process).
4. All processes use a single shared memory to write the different file part
5. Once all parts are downloaded the main process save it to the final file

Implement the client (under Linux)

a) You should use sockets for client/client communication
b) Only implement the client part responsible for the steps above
c) Keep your final code in t he specified area

Good Luck

name	Maher
id	

60

```
#define key 200
void downloadPart(int Acksize, int id, int size, char *IP, int port);
int getTotalsize(FILE* f, int N);

int main( ){
    char fileName[20] = "file.txt";
    getfileInfo(fileName);                               5
    FILE * f= fopen(INFOFILE, "r");
    int totalsize= getTotalSize(
    int N; fscanf(f, "%d", &N);                          10
    int totalSize= getTotal Size(f, N);       } get total size of file
    //Now creating shared Memory
    int shm-id= shmget(KEY, total Size, 0);   } create shared memory
char* b= shmat(shm-id, NULL, 0);                and attached to it
    int cumSum=0 ;//Base address for each part (comulative sum)
    f= fopen(INFOFILE, "r")
    fscanf(f,"%d, &N);
    int i;
    for(i=0 ; i<N ; i++){ //traverse each part
        int id, port, size;                              5
        char IP[20]
        scanf(f, "%d%d%s%d", &id, &size, IP, &port);
        if(!fork())
        {
            downloadPart(cumSum, id, size, IP, port);


        }
        else
        AckSum += size; // base address for next process

}
```

يتبع في الورقة المقبلة

```
int getTotalSize(FILE *f, int N){
    int i, sum=0;
    for(i=0; i<N; i++){
        int id, size, port;
        char IP[20];
        fscanf(f, "%d %d %s %d", &id, &size, IP, &port);
        sum += size;
    }
    fclose(f);
    return sum;
}
```

(5)

main 내에 작성

```
int i
for(i=0; i<N; i++)
    wait(NULL); //wait until them to finish

char *totalFile = (char*) malloc(totalsize * sizeof(char));

strcpy(totalFile, b);   // base address
```

(5) copy to string

```
FILE *outfile = fopen("outfile", "w");
fprintf(outfile, "%s", totalfile);
fclose(outfile);
```

print shared memory to file

```
shmdt(b);                       } detach
shmctl(shm_id, SHM_RM, NULL);   } remove shared memory

return 0;
}
```

int shm-id = shmget(KEY, total size, 0)

char *b = shmat(shm-id, NULL, 0); | char *org = b; // original base address

b += cum size; // move to its location

} attach to shared Memory

~~char *org = b;~~

Void downloadPart(int cum size, int id, int size, char *IP, int port){

int fd = socket(Af_INET, SOCK_STREAM, 0);

struct sockaddr_in peer;

peer.sin_family = Af_INET
peer.sin_port = htons(Port);
peer.sin_addr.s_addr = Inet_addr(IP);

} create sockaddr_in object

if ((connect(fd, (struct sockaddr *)&peer, sizeof(peer)) < 0){
perror("connect");
exit(1);
}

} connecting

if ((write(fd, &id, sizeof(id)) < 0)
perror("write");
exit(2);
}

5

16

} send id to peer

char Buff[1024]
while (size > 0) { // read until the size finishes

if (numBytes = read(fd, Buff, 1024)) < 0){
perror("read");
exit(3);
}

size -= numBytes;

sprintf(b, "%s", buff);

15

b += numbytes; // shift shmemmy address

} read data and store it to shared memory

shmdt(org);
close(fd);
}

} deattach and close file

Signals & Processes

typedef void (*sighandler_t)(int);

sighandler_t **signal**(int *signum*,
sighandler_t *handler*);

int **raise**(int sig);

int **kill**(pid_t *pid*, int *sig*);

pid_t **getpid**(void);

pid_t **getppid**(void);

pid_t **fork**(void);

pipes

int **pipe**(int* fd);

int ***close***(int fd*);*

Int **read**(int fd ,void* buf, int bufsize);

Int **write**(int fd ,void* buf, int bufsize);

fifo

int mknod(char *pathname, mode_t mode, dev_t dev);

int fopen(FIFO_name, "rw");

int fclose(int fd);

Message queue

int **msgget** (key_t key, int msgflg);

int **msgsnd** (int msqid, struct msgbuf *msgp, int msgsz, int msgflg);

int **msgrcv** (int msqid, struct msgbuf *msgp, int msgsz, long mtype, int msgflg);

int **msgctl**(int msgqid, int cmd, struct msqid_ds *buf);, IPC_RMID

struct msgbuf {

long mtype; /* type of message */

char mtext[1]; /* message data*/

};

Shared memory

int **shmget** (key_t key, int size, int shmflg);

int **shmat** (int shmid, char *shmaddr, int flg);

int **shmctl** (int shmqid, int cmd, struct shmid_ds *buf);

int **shmdt** (char *shmaddr);

Semaphore

int semget (key_t key, int nsems, int semflg);

int semop (int semid, struct sembuf *sops, unsigned nsops);

int semctl (int semid, int semnum, int cmd, union semun arg);

union semun {

int val; /* value for SETVAL */

struct semid_ds *buf; // buffer for IPC_

ushort *array; // array for GETALL & SETALL

struct seminfo *__buf; // buffer for IPC_INFO

void *__pad; };

struct sembuf {

ushort sem_num;

short sem_op;

short sem_flg; };

acquire = {0, -1, SEM_UNDO},

release = {0, 1, SEM_UNDO};

sockets

```
fd = socket(AF_INET, SOCK_STREAM, 0);

bind(fd, (struct sockaddr*) &srv, sizeof(srv));

listen(fd, 5);

accept(fd, (struct sockaddr*) &cli, &cli_len);

connect(fd, (struct sockaddr*) &srv, sizeof(srv))
```

```
struct sockaddr_in {
u_char sin_family; /* Address Family */
u_short sin_port; /* UDP or TCP Port# */
/* network byte ordered */
struct in_addr sin_addr; /* Internet Address */
char sin_zero[8]; /* unused */

};
```

```
nbytes = recvfrom(fd, buf,
sizeof(buf), 0 ,(struct sockaddr*)
cli, &cli_len);

sendto(fd, buf, sizeof(buf), 0 ,
(struct sockaddr*) &srv, sizeof(srv));

unsigned long htonl(unsigned long hostlong);
unsigned short htons(unsigned short hostshort)
unsigned long ntohl(unsigned long netlong);

unsigned short ntohs(unsigned short netshort);
```

Threads

```
int pthread_create (pthread_t * id, const pthread_attr_t *attrib ,
void (*thread_fun)(void*), void *arguments);

pthread_join (pthread_t thread, void **status)

pthread_exit( );

pthread_mutex_t lock;

pthread_mutex_lock( &lock ) ;

pthread_mutex_unlock( &lock )

pthread_cond_t SpaceAvailable;

pthread_cond_init (&SpaceAvailable, NULL );

pthread_cond_wait (&condition, &mutex)

pthread_cond_signal(&condition)

pthread_cond_broadcast(&condition)
```