UTN FRD SO – SEGUNDO PARCIAL (T1)

1: ¿Por qué razón la asignación encadenada o enlazada no tendría buen rendimiento sobre un archivo de acceso directo?

2: ¿Cuál es la salida de esta porción de código cuando es ejecutado?

```
int p1[2];pipe(p1);pid_t pid = fork();
if ( pid == 0 ) {   char c; char tmp[20]; close(p1[1]); int t=0,n=1,i=0;
        while(n) {
                n = read(p1[0],&c,1);
                if ( n ) { if ( c >= '0' && c <= '9' ) tmp[i++]=c;
                        else { tmp[i]='\0'; t+=atoi(tmp); i=0; }
                }
        } close(p1[0]);printf("%d\n",t);
} else {  close(p1[0]);int n=100;char s[100];
        while(n<=300) {
                snprintf(s,100,"%d",n);   write(p1[1],s,strlen(s)+1); n+=100;
        } close(p1[1]); wait(0);
} return 0;
```

3: Un proceso productor usa un buffer circular en memoria compartida (0xA, ya está creada) de 320 bytes, cada producción ocupa 32bytes. Existe un único proceso consumidor. La función **`void produce(char *to)`** produce 32 bytes en formato string (incluye \0) y lo copia en **`to`**, el consumidor solo imprime en pantalla el string consumido. Todo finaliza cuando se produce el mensaje “chau”. Debe indicar los semáforos que necesita, sus valores iniciales, se asume que los mismos están creados e inicializados, cuenta con las funciones **`SemWait(semid,nro sem)`** y **`SemSignal(semid,nro sem)`** *solo complete el código principal del proceso consumidor*. Le damos como ayuda una parte incompleta y errónea del código del productor:

```
int shmid = shmget(0xA,0,0), salir=0;
char *p = (char *) shmat(shmid,0,0);
do { produce(p); if ( strncmp(p,"chau",4) == 0 ) salir=1;
     p+=32;
} while(!salir);
```

UTN FRD SO – SEGUNDO PARCIAL (T1)

1: ¿Por qué razón la asignación encadenada o enlazada no tendría buen rendimiento sobre un archivo de acceso directo?

2: ¿Cuál es la salida de esta porción de código cuando es ejecutado?

```
int p1[2];pipe(p1);pid_t pid = fork();
if ( pid == 0 ) { char c; char tmp[20]; close(p1[1]); int t=0,n=1,i=0;
        while(n) {
                n = read(p1[0],&c,1);
                if ( n ) { if ( c >= '0' && c <= '9' ) tmp[i++]=c;
                        else { tmp[i]='\0'; t+=atoi(tmp); i=0; }
                }
        }  close(p1[0]);printf("%d\n",t);
} else {  close(p1[0]);int n=100;char s[100];
        while(n<=300) {
                snprintf(s,100,"%d",n);   write(p1[1],s,strlen(s)+1); n+=100;
        }  close(p1[1]); wait(0);
} return 0;
```

3: Un proceso productor usa un buffer circular en memoria compartida (0xA, ya está creada) de 320 bytes, cada producción ocupa 32bytes. Existe un único proceso consumidor. La función **`void produce(char *to)`** produce 32 bytes en formato string (incluye \0) y lo copia en **`to`**, el consumidor solo imprime en pantalla el string consumido. Todo finaliza cuando se produce el mensaje “chau”. Debe indicar los semáforos que necesita, sus valores iniciales, se asume que los mismos están creados e inicializados, cuenta con las funciones **`SemWait(semid,nro sem)`** y **`SemSignal(semid,nro sem)`** solo complete el código principal del proceso consumidor. Le damos como ayuda una parte incompleta y errónea del código del productor:

```
int shmid = shmget(0xA,0,0), salir=0;
char *p = (char *) shmat(shmid,0,0);
do {     produce(p); if ( strncmp(p,"chau",4) == 0 ) salir=1;
     p+=32;
} while(!salir);
```

Respuestas Posibles:

1: ¿Por qué razón la asignación encadenada o enlazada no tendría buen rendimiento sobre un archivo de acceso directo?
Porque ir al i-esimo bloque implica recorrer la lista enlazada hacia adelante a partir de un bloque inicial y así sucesivamente para cada petición. Obtener el i-esimo bloque puede implicar la lectura de muchos bloques.

2: 600\n

(El proceso padre escribe en el pipe: "100\0200\0300\0" el proceso hijo lee de a un carácter y los concatena en tmp, cuando encuentra \0 pasa tmp a entero y lo acumula en t. Por lo tanto, t=100+200+300=600)

3: Un proceso productor usa un buffer circular en memoria compartida (0xA, ya está creada) de 320 bytes, cada producción ocupa 32bytes. Existe un único proceso consumidor. La función **`void produce(char *to)`** produce 32 bytes en formato string (incluye \0) y lo copia en **`to`**, el consumidor solo imprime en pantalla el string consumido. Todo finaliza cuando se produce el mensaje "chau". Debe indicar los semáforos que necesita, sus valores iniciales, se asume que los mismos están creados e inicializados, cuenta con las funciones **`SemWait(semid,nro sem)`** y **`SemSignal(semid,nro sem)`** solo complete el código principal del proceso consumidor. Le damos como ayuda una parte incompleta y errónea del código del productor:

```
int shmid = shmget(0xA,0,0), salir=0;
char *p = (char *) shmat(shmid,0,0);
do {     produce(p); if ( strncmp(p,"chau",4) == 0 ) salir=1;
     p+=32;
} while(!salir);
```

Necesita 2 semáforos,
S0=10 semaforo productor
S1=0 semaforo consumidor
Codigo del productor (Tema II)

```
int shmid = shmget(0xA,0,0);
int semid = semget(0xA,0,0);
int salir=0;
char *p = (char *) shmat(shmid,0,0);
char *pinicial = p;
char *ptope = p+320;
do {
     semWait(semid,0); // P(S0)
          produce(p);
          if ( strncmp(p,"chau",4) == 0 ) salir=1;
          p+=32;
          if ( p >= ptope ) p = pinicial;
     semSignal(semid,1); // V(S1);
} while(!salir);
shmdt(pinicial);
```

Codigo del consumidor (Tema I)

```
int shmid = shmget(0xA,0,0);
int semid = semget(0xA,0,0);
int salir=0;
char *p = (char *) shmat(shmid,0,0);
char *pinicial = p;
char *ptope = p+320;
do {
     semWait(semid,1); // P(S1)
          printf("%s\n",p);
          if ( strncmp(p,"chau",4) == 0 ) salir=1;
          p+=32;
          if ( p >= ptope ) p = pinicial;
     semSignal(semid,0); // V(S0);
} while(!salir);
shmdt(pinicial);
```

Ejercicios de Juan:

2) Para la siguiente secuencia de ejecución de procesos: (PA o PB) (PC y PD) y así continúan. Se Pide: Mínima cantidad de semáforos binarios. Inicialización de los semáforos. Sección de entrada y sección de salida en cada proceso.

Secuencias validas
ACD
ADC
BCD
BDC
Semaforos requeridos
Scab=1
Sdab=1
Sc=0
Sd=0

	A	B	C	D
Sec. Entrada	P(Scab) P(Sdab)	P(Scab) P(Sdab)	P(Sc)	P(Sd)
Sec. Critica	SC	SC	SC	SC
Sec. Salida	V(Sc) V(Sd)	V(Sc) V(Sd)	V(Scab)	V(Sdab)

Probado con autosem, ver archivo AoB_CDoDC.txt y log.txt

Lo mismo, con otros nombres:
Sa = 1
Sb = 1
Sc = 0
Sd = 0

	A	B	C	D
Sec. Entrada	P(Sa) P(Sb)	P(Sa) P(Sb)	P(Sc)	P(Sd)
Sec. Critica	SC	SC	SC	SC
Sec. Salida	V(Sc) V(Sd)	V(Sc) V(Sd)	V(Sa)	V(Sb)

3) Dado el siguiente proceso, en un sistema de paginación bajo demanda, las direcciones lógicas son de 16 bits y la página es de 4 KBytes y los frames libres son los siguientes 7, A, D, 9

Proceso	Responda:

MOV AX,[212A]
MOV BX,[145B]
MOV CX,[378C]
ADD AX,BX
MOV [712F],AX
	Por cada acceso a memoria a datos mapee la dirección lógica a dirección física

Dirección lógica=16 bits=4 digitos hexa (por cada digito hexa necesito 4 bits: 0-15)
Offset=Tamaño página=Tamaño frame=4 KBytes=12 bits (0 – 4095 dec)
Dirección lógica=[nro. de página – 4 bits = 0-15 dec = 0-F hexa = 1 digito hexa][desplazamiento – 12 bits = 0-4095 dec = 000-FFF hexa = 3 digitos hexa]

Acceso	**Dirección Lógica**	**Dirección Física**
212A	[2][12A]	[7][12A]
145B	[1][45B]	[A][45B]
378C	[3][78C]	[D][78C]
712F	[7][12F]	[9][12F]

	Tabla de páginas para este proceso
0	
1	A
2	7
3	D
4	
5	
6	
7	9

Atte. Guillermo Cherencio.