1: Realice la gráfica de la estructura de procesos generada por el siguiente código, en la gráfica asigne PID a cada proceso. (2)

```
int main(void) {
      pid_t pid;   pid = fork();   pid = fork();    pid = fork();
      if (pid > 0) fork();
      return pid ;
}
```

2: Modifique el siguiente código para que no exista ningún proceso huérfano ni zombie. (2.5)

```
int main(void) {
   pid_t pid;  pid = fork();
   if ( pid > 0 )  {
      pid = fork();
      if ( pid > 0 )  {
         pid = fork();
         if ( pid > 0 ) pid = fork() ;
     }
   }
   return 0;
}
```

3: Complete el siguiente programa para que el padre repita en pantalla “Padre dice hola” durante 5 segundos y luego terminen el padre y el hijo. (Usar señales, no usar sleep). (2.5)

```
void  handler(void);
int main(void) {
   pid_t pid ;
   pid = fork();
   if (pid > 0)    { ..... }
   else    { ..... }
   return 0 ;
}
void handler(void) {   printf("Padre dice hola\n"); }
```

4: Escriba en pseudocódigo la siguiente sincronización BACABACABACA….. de ejecución de la sección crítica entre procesos, nombre los semáforos, indique valor inicial y escriba la sección de entrada y la sección de salida de cada proceso. (3)

Criterio de Aprobación: obtener -al menos- 6 puntos sobre un total de 10.

Teoría:

1. En un sistema de procesamiento batch o por lotes, ¿Qué sucede cuando un programa del usuario ejecuta una instrucción privilegiada (asumiendo que el hardware tiene la facilidad de contar con este tipo de instrucciones)? ¿Cuál cree Ud. que es la razón de ello? (2)
2. ¿Cuáles son las características de un proceso? (2)
3. ¿Cuáles son los componentes de un proceso? (2)
4. ¿Por qué razón se realiza un cambio de proceso (process switching)? (2)
5. ¿Por qué un cambio de contexto entre procesos es más “pesado o lento” que un cambio de contexto entre hilos? (2)

Criterio de Aprobación: obtener -al menos- 6 puntos sobre un total de 10.

Criterio de Calificación:
6 p = 6 (acorde a exactitud) 7 p = 7 8 p = 8 9 p = 9 10 p = 10 (acorde a exactitud)

Practica

1: Realice la gráfica de la estructura de procesos generada por el siguiente código, en la gráfica asigne PID a cada proceso. (2)

```
int main(void) {
      pid_t pid;   pid = fork();   pid = fork();    pid = fork();
      if (pid > 0) fork();
      return pid ;
}
```

```
3018 (shell)
  3204
    3205
      3211
        3214
        3215
      3213
      3212
    3206
      3209
      3210
    3207
    3208
```

2: Modifique el siguiente código para que no exista ningún proceso huérfano ni zombie. (2.5)

```
int main(void) {
     pid_t pid;
     pid = fork();
     if ( pid > 0 )  {
          wait(0);
          pid = fork();
          if ( pid > 0 )  {
               wait(0);
               pid = fork();
               if ( pid > 0 ) {
                    wait(0);
                    pid = fork();
                    if ( pid > 0 ) wait(0);
               }
          }
     }
     return 0;
```

```
}
```

3: Complete el siguiente programa para que el padre repita en pantalla “Padre dice hola” durante 5 segundos y luego terminen el padre y el hijo. (Usar señales, no usar sleep). (2.5)

```
void handler(void);
void handler2(int);
int loop=1;
int main(int argc, char **argv) {
     pid_t pid;
     pid = fork();
     if (pid > 0) {
          signal(SIGUSR1,handler2);
          while(loop) handler();
          wait(0);
     } else {
          signal(SIGALRM,handler2);
          alarm(5);
          pause();
          kill(getppid(),SIGUSR1);
     }
     return 0;
}

void handler(void) {
     printf("Padre dice hola\n");
}

void handler2(int signo) {
     loop=0;
}
```

```
void handler(void);
void handler2(int);
int loop=1;
int main(int argc, char **argv) {
     pid_t pid;
     pid = fork();
     if (pid > 0) {
          signal(SIGALRM,handler2);
          alarm(5);
          while(loop) handler();
          wait(0);
     } else {
          alarm(5);
          pause(); // while(loop);
     }
     return 0;
}

void handler(void) {
```

```
     printf("Padre dice hola\n");
}

void handler2(int signo) {
     loop=0;
}
```

```
void handler(void);
void handler2(int);
int loop=1;
int main(int argc, char **argv) {
     pid_t pid;
     pid = fork();
     if (pid > 0) {
          signal(SIGCHLD,handler2);
          while(loop) handler();
          wait(0);
     } else {
          alarm(5);
          pause(); // while(loop);
     }
     return 0;
}

void handler(void) {
     printf("Padre dice hola\n");
}

void handler2(int signo) {
     loop=0;
}
```

4 Escriba en pseudocódigo la siguiente sincronización BACABACABACA….. de ejecución de la sección crítica entre procesos, nombre los semáforos, indique valor inicial y escriba la sección de entrada y la sección de salida de cada proceso. (3)

Sa=0 Sb=1 Sc=0		
A	B	C
wait(Sa) SC signal(Sc) wait(Sa) SC signal(Sb)	wait(Sb) SC signal(Sa)	wait(Sc) SC signal(Sa)

O bien:

Sa = 0 Sb = 1 Sc = 0 Sx = 1		
A	B	C
wait(Sa) SC Signal(Sx)	wait(Sb) wait(Sx) SC Signal(Sc) Signal(Sa)	wait(Sc) wait(Sx) SC Signal(Sb) Signal(Sa)

Teoría

1 En un sistema de procesamiento batch o por lotes, ¿Qué sucede cuando un programa del usuario ejecuta una instrucción privilegiada (asumiendo que el hardware tiene la facilidad de contar con este tipo de instrucciones)? ¿Cuál cree Ud. que es la razón de ello?
Si un programa de usuario pretende ejecutar una instrucción privilegiada se producirá un error, puesto que este tipo de instrucciones no esta permitido para un programa de usuario. Solo el monitor puede ejecutar instrucciones privilegiadas puesto que éstas tienen mayor prioridad que las instrucciones del usuario, si los programas de usuario pudieran ejecutar estas instrucciones habría -al menos- dos problemas: 1. si un programa de usuario acaparara todo el tiempo de cpu ejecutando instrucciones privilegiadas (de mayor prioridad), el monitor no podría tomar el control. 2. se perdería la aislación / abstracción del hardware que provee el monitor, la interacción con el hardware debe hacerse a través del monitor y no directamente desde los programas de usuario.

2 ¿Cuáles son las características de un proceso?
-Programa en ejecución
-"espíritu animado" de un programa
-entidad asignada a CPU y ejecutada por ésta
-instancia de programa en ejecución
-unidad de actividad caracterizada por la ejecución de una secuencia de instrucciones, un estado y un conjunto de recursos asociados
-entidad formada por n elementos (codigo del programa, conjunto de datos, PCB (process control block))
-entidad que describe un comportamiento o taza, listado de la secuencia de instrucciones del proceso

3 ¿Cuáles son los componentes de un proceso?
-Programa ejecutable
-Datos asociados al programa (variables, espacio de trabajo, buffers, etc.)
-Contexto de Ejecución: info. necesaria para que el SO pueda administrar el proceso, contenido de los registros del procesador, PC, registros de datos, prioridad, etc.

4 ¿Por qué razón se realiza un cambio de proceso (process switching)?
Pag.138. se realizar porque ocurre una interrupción (causa externa al proc. en ejecucion), ocurre un trap (causa interna al proc. en ejecución), se realiza una llamada al SO (system call).

5 ¿Por qué un cambio de contexto entre procesos es más "pesado o lento" que un cambio de contexto entre hilos?

Porque para cambiar entre procesos se requiere la intervención del SO para que éste guarde el contexto de ejecución de proceso actual, lo descargue, recupere el contexto de ejecución del nuevo proceso a ejecutar y lo ponga en ejecución. Mientras que un hilo o thread comparte gran parte del contexto del proceso que lo contiene, por lo tanto se puede cambiar de un hilo a otro dentro del mismo proceso sin necesidad de intervención del SO (tampoco requiere un cambio de modo) y sólo salvando y restaurando el contexto propio del thread, véase Fig. 4.2. Los hilos comparten el bloque de control de proceso, solo hay que salvar/restaurar el bloque de control de hilo.