CC3301 Programación de Software de Sistemas

Examen – Semestre Primavera 2013 – Prof.: Luis Mateu

Pregunta 1

i. Programe la función *techoMod8* que recibe un entero *n* y entrega el menor entero múltiplo de 8 que es mayor o igual a *n*. *Restricción:* Ud. no puede usar la multiplicación, división o resto. *Hint:* sume 7 y luego lleve a 0 los 3 bits menos significativos usando los operadores de bits.

ii. Programe la función *doble_malloc* que pide memoria para 2 objetos. El código de la izquierda muestra un ejemplo de uso. *Restricción:* Ud. puede invocar malloc *una sola vez*. Tanto *px* como *pn* deben estar alineados a 8 bytes (la dirección debe ser múltiplo de 8). Recuerde que malloc entrega una dirección alineada a 8 bytes.

```
float *px;  Nodo *pn;
doble_malloc(
  (void**)&px, sizeof(float),
  (void**)&pn, sizeof(Nodo));
pn->prox= NULL; pn->pdata= px;
```

iii. Programe la función *borrarMasIzq* que elimina el nodo de más a la izquierda de un árbol de búsqueda binaria. El encabezado de la función es:

```
void borrarMasIzq(Nodo **p_raiz);
```

La raíz del árbol se pasa por referencia porque la raíz va a ser eliminada si es el nodo de más a la izquierda. En ese caso en `*p_raiz` se entrega la nueva raíz del árbol. *Restricción:* no puede usar ciclos como *while*. Use recursividad.

Pregunta 2

Considere nuevamente el problema de la búsqueda de un factor para un número *x* de gran tamaño (control 3).

Parte a.- (1 punto) Discuta si la siguiente implementación de *factorizar* es correcta:

```
uint factorizar(ulonglong x) {
  uint sup= (uint)sqrt((double)x);
  uint fct_h, fct_p;
  int pid= fork();
  if (pid==0) {
    fct_h= buscarFactor(x, 2, sup/2);
    exit(0);
  } else {
    fct_p= buscarFactor(x, sup/2+1, sup);
    waitpid(pid, NULL, 0);
  }
  return fct_h!=0 ? fct_h : fct_p;
}
```

En donde *buscarFactor(x, i, j)* explora factores en el rango [*i*, *j*], retornando el factor encontrado o 0 en caso de fracaso.

Parte b.- (2.5 puntos) Reprograme la función *factorizar* para que ambas llamadas a *buscarFactor* se realicen en subprocesos. Para ello cree 2 subprocesos hijos con *fork*, los que deben entregar sus resultados al padre usando pipes. El proceso padre solo se sienta a esperar los resultados leyendo primero un pipe y luego el otro.

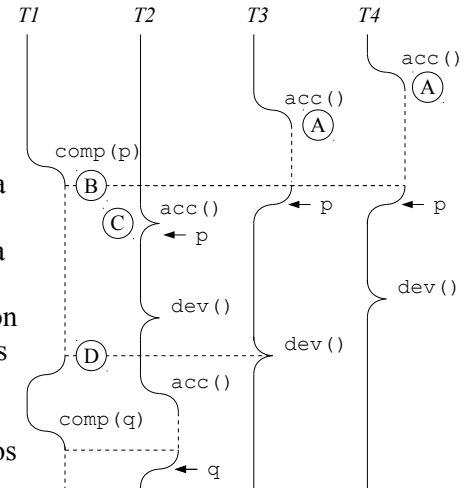
Parte c.- (2.5 puntos) Si uno de los subprocesos envía su resultado, ya no necesita esperar el resultado del otro subproceso. Reprograme la función de la parte b.- de modo que el padre use *select* para leer el pipe del subproceso que envíe primero su resultado. Luego mate el otro subproceso y retorne de inmediato el factor encontrado.

Pregunta 3

Programe las siguientes funciones cuyo fin es permitir que varios threads compartan datos en modo lectura:

- `void compartir(void *ptr)`: Ofrece compartir los datos apuntados por *ptr* con los threads que llamen a *acceder*. *Compartir* queda en espera hasta que los threads notifiquen que desocuparon los datos llamando a *devolver*.
- `void *acceder( )`: Solicita acceso a los datos ofrecidos con *compartir*. Si hay una llamada a *compartir* en espera, retorna de inmediato el puntero *ptr* suministrado mediante *compartir*. Si no, espera hasta la próxima invocación de *compartir*.
- `void devolver( )`: Notifica que los datos compartidos ya no se usarán.

El diagrama de la derecha explica el funcionamiento pedido. En A *acceder* se bloquea hasta que otro thread invoque *compartir*. Esto ocurre en B, lo que hace que todos los threads que esperaban en una llamada a *acceder* se desbloqueen retornando el puntero a los datos (p en este caso). La llamada a *compartir* queda en espera hasta que todos los threads que llamaron a *acceder* notifiquen que no usarán más los datos invocando *devolver*. En C, como hay una llamada a *compartir* en espera, *acceder* retorna de inmediato los datos compartidos. En D se invoca el último *devolver*, y por lo tanto *compartir* retorna. Si se invoca *compartir* y no hay threads que llamaron a acceder, *compartir* retorna de inmediato.



Para la sincronización use un *mutex* y una condición de *pthreads*, ambos almacenados en variables globales.