CC3301 Programación de Software de Sistemas

Control 2 – Semestre Otoño 2013

Prof.: Luis Mateu

Pregunta 1

Parte a.- Implemente una cola de prioridades usando *listas simplemente enlazadas (4 puntos)*. La cola de prioridad posee las siguientes operaciones:

```
typedef int (*CompPri)(void *a, void *b);
ColaPri *nuevaColaPri(CompPri compPri);
void agregar(ColaPri *cola, void *a);
void *extraer(ColaPri *cola);
int vacia(ColaPri *cola);
```

En donde *nuevaColaPri* es el constructor y recibe un puntero a una función que compara las prioridades de dos elementos de la cola (*compPri*). Esta función entrega -1 si *a* tiene mejor prioridad que *b*, 0 si es la misma prioridad o 1 si es peor. La función *agregar* agrega el elemento *a* a la cola de prioridad *cola*. La función *extraer* extrae y entrega el elemento de mejor prioridad entre todos los elementos de la cola. La función *vacia* entrega 1 si la cola está vacía y 0 si no.

Este es un ejemplo de uso de la cola de prioridades:

`CompPri cmp= (CompPri)strcmp;`	el comparador
`ColaPri c= nuevaColaPri(cmp);`	entrega una nueva cola
`char *r, *s, *t;`	
`agregar(c, "que");`	agrega un elemento
`if (vacia(c)) ...`	entrega 0 (falso)
`agregar(c, "hola");`	
`r= (char*)extraer(c);`	entrega "hola"
`agregar(c, "tal");`	
`s= (char*)extraer(c);`	entrega "que"
`t= (char*)extraer(c);`	entrega "tal"
`if (vacia(c)) ...`	entrega 1 (verdadero)

Parte b.- Escriba un programa que entregue los parámetros recibidos en la línea de comandos ordenados numéricamente (2 puntos). Ejemplo de uso:

```
% ordenar 24 3 45 78 -12
-12 3 24 45 78
```

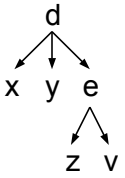
Para el ordenamiento Ud. ***debe*** usar la cola de prioridades de la ***parte a.-*** Recuerde que la función *atoi* le permite convertir de un *string* al valor numérico que representa ese string.

Pregunta 2

Parte i.- Escriba la función *lsRec* que dado un string que representa un archivo o directorio, entrega en cualquier orden y a través de un fd (*file descriptor*) los archivos accesibles a partir de ese string. El encabezado de la función es:

```
void lsRec(char *path, int fd);
```

Por cada archivo accesible desde *path* Ud. debe escribir en fd 2 bytes que indican el largo del *path* de acceso a ese archivo seguido de su *path* de acceso.

jerarquía de archivos	***ejemplo de llamada***	***largo del path***	***paths accesibles escritos en fd***
	`lsRec("d/e/v", 10)`	5	`d/e/v`

	`lsRec("d", 10)`	3
		3
		5
		5
			`d/x`
			`d/y`
			`d/e/z`
			`d/e/v`

Parte ii.- Implemente un iterador para los archivos accesibles desde un archivo o directorio. El iterador consiste en estas 3 funciones:

```
Iterador *nuevoIterador(char *path);
char *prox(Iterador *it);
void *destruirIterador(Iterador *it);
```

Este es un ejemplo de uso considerando la jerarquía de archivos de la ***parte a.-***

```
char *p1, *p2, *p3, *p4, *p5;
Iterador *it= nuevoIterador("d");
p1= prox(it);                                  entrega "d/x"
p2= prox(it);                                  entrega "d/y"
p3= prox(it);                                  entrega "d/e/z"
p4= prox(it);                                  entrega "d/e/v"
p5= prox(it);                                  entrega NULL
destruirIterador(it);
free(p1); free(p2); free(p3); free(p4);
```

Restricción: Resuelva este problema de la siguiente manera. En la función *nuevoIterador* invoque fork para crear un proceso hijo que haga el recorrido recursivo (con *lsRec*) de la jerarquía de archivos. Suponga que un *path* no excede los 1024 bytes. Use un *pipe* para comunicar el padre con el hijo. En el padre, la función *prox* obtiene los archivos accesibles leyendo del *pipe*. Invoque malloc para obtener espacio para cada *path*. En la función *destruirIterador* preocúpese de enterrar adecuadamente el proceso hijo.