Apuntes de PSP

* Los includes que hemos visto son:
  + #include <stdio.h>
  + #include <strings.h>
  + #include <unistd.h>
  + #include <stdlib.h>
  + #include <signal.h>
  + #include <sys/types.h>
  + #include <sys/wait.h>
  + #include <math.h>
* Para comparar chars lo haces de esta manera: if(variableQueContieneChar==’\*’){

Printf(“Soy un mas”);

}

* Para declarar finales es:
  + #define nombreVariable 0-9/strings/double/….
    - Ejemplo: #define CERO 0
    - Ejemplo #define APUNTES “APUNTES”
* Para crear un random es:
  + Variable=rand%el numero que quieras;
  + Para hacer que sea random, se usa srand(time(NULL))
* Para crear un pipe es:
  + Creas un array llamado acciones[2];
  + Pipe(acciones)
    - Si quiero cerrar un extremo es: close(acciones[0]) si quieres cerrar lectura y close(acciones[1]) si quieres cerrar la escritura
    - Si quiero hacer lectura o escritura: read(acciones[0](lectura),variable a pasar, sizeof(variable a pasar)). Write(acciones[1],variable para enviar, sizeof(Variable para enviar));
    - TEN EN CUENTA QUE TIENEN QUE TENER EL MISMO TAMAÑO SI ES UN ARRAY Y EL MISMO TIPO
    - IMPORTANTE, PARA PASAR CADENAS DE TEXTO Y/O CARACTERES, CUANDO DEFINAS UN pipe(variableQueContieneUnChar);
    - CUANDO LO ESCRIBAS CON WRITE VA DE ESTA MANERA: write(envioSimbolo[1],&variableQueContieneUnChar,sizeof(variableQueContieneUnchar)) Mas detalles en: REPASOEXAMEN2/ejercicio6.c
* Escritura en archivos se hace de esta manera:
  + Primero definimos el archivo : FILE\* nombreArchivo;
  + Luego lo abrimos: archivo=fopen(“nombreDelArchivo.txt”,”\*”);
  + \*hay varios modos para abrirlo:
    - R: lectura
    - W: escritura
    - A: append y no sobrescribe
    - R+/W+: lectura y escritura
    - A+: escritura y no sobrescribe
  + Si quiero escribir en el documento se puede hacer de dos maneras:
    - Fprintf(archivo,”lo que haya que escribir”)
    - Fwrite(datos, sizeof(datos),archivo)
    - IMPORTANTE NUNCA HACER: ~~archivo~~=fprintf/fwrite (archivo,””);
  + IMPORTANTE, CUANDO SE HAYA TERMINADO HACER fclose(archivo)
* Para crear y manejar las señales como dioses del olimpo necesitas:
  + Para cualquier señal que necesites para manejar determianda señal haces:
    - Signal(Tipo de señal que quieres manejar,función que va a manejar esa señal).
    - Si estas usando SIGNKILL necesitas hacer:
      * Kill(lo que quieras matar, SIGKILL)
    - IMPORTANTE!!!! Kill es un comando para mandar señales, es decir, si quieres “Automatizar señales” tienes que lanzar un kill a la variable del hijo o a lo que tenga pid\_t y luego el tipo de señal que quieras
    - Kill(pid\_t hijoX,señalALanzar);
    - mandar, si esto lo combinas con Signal, puedes lanzar señales y encima personalizar esas Señales, funciona como los try/catch en java
    - Los que hemos visto son:
      * SIGKILL: para matar
      * SIGUSR1/SIGUSR2: el usuario 1 tiene que realizar una acción
      * SIGINT
* Fflush(stdout): es una función que lo que hace es forzar todos los printf posteriores a este para que se escriban en la pantalla de ejecución del proceso
* En los printf tenemos:
  + %d para los integer
  + %s para los strings/char
  + %c para chars
* Para comparar cadenas se usa el strcmp() de esta manera: int resultado=strcmp(cadena1,cadena2); ya que devuelve 1 si es desigual y 0 si es igual
* Para ordenar números de mayor a menor, se hace de esta manera:
  + Qsort(array de números, tamaño del array, sizeof(int), función comparar numeros);

**Funciones**

* int compararNumeros(const void \*a, const void \*b) {
* return (\*(int \*)b - \*(int \*)a);
* }

int esPrimo(int num) {

if (num < 2) return 0;

for (int i = 2; i \* i <= num; ++i) {

if (num % i == 0) return 0;

}

return 1;

}

int generarNumero(int longitud) {

int rangoInicial = 1;

int rangoFinal = 1;

for (int i = 1; i < longitud; ++i) {

rangoInicial \*= 10;

rangoFinal \*= 10;

}

rangoFinal = rangoFinal \* 10 - 1;

return rand() % (rangoFinal - rangoInicial + 1) + rangoInicial;

}

La de generar numero es para hacer cuando se pasa por argumento un numero x que cree números con un intervalo, si pongo 2 va a ir del 10-99 si pongo 3, del 100-999 y asi sucesivamente

Apuntes de java

* Para empezar, siempre tienes que crear una instancia del método que vas a hacer las acciones con ese Thread: por ejemplo
  + Tenemos un método main y otro llamado ejercicio1, si quiero crear ejecutarlo usando Threads tengo que:
    - Primero creamos la instancia: ejercicio1 holamundo=new holamundo;
    - Holamundo.start();
  + En ejercicio1 tenemos que extender de Thread:
    - Public class ejercicio1 extends Thread{
      * Y como estamos llamando a start creamos start():public void start(){sout(“Hola mundo”)}
    - }
* Si vamos a hacer uso de la implementación de Runnable tenemos que:
  + Tenemos un método main y otro método llamado ejercicio2, primero instanciamos:
    - Ejercicio2 mundo2=new ejercicio2();
  + Luego creamos un thread:
    - Thread contendorMundo2=new Thread(mundo2);
  + Muy importante meterlo dentro porque sino no lo llamamos nunca:
    - contenedorMundo2.run();
  + en ejercicio2 implementamos Runnable y haciendo uso de VSCODE se nos crea la clase run que es la que va a ejecutar el main
* Si vamos a hacer uso de los lambda tenemos que:
  + No es necesario crear ningún método ni instancia del mismo, podemos hacerlo todo junto:
    - Dentro del main: Thread mundo3=new Thread(()->sout(“Mundo3”));
    - Y luego lo iniciamos: mundo3.start();
* Para crear alias de los hilos es necesario hacer al thread: thread.setName(“hilo”);
* Para los bloques que se crean de syncronize(){} siempre entre los paréntesis, tiene que haber el objeto lock ya que como actúa de cerrojo el syncronize, es siempre bueno hacerlo.
  + Primero el objeto lock: public static final Object lock=new Object() quedando asi: syncronize(claseMain.lock){} y si lo creamos en el main el bloque syncronize() tan solo seria: syncronize(lock){} esto se hace para que, al usar el mismo objeto este se puede estar comunicando entre si para saber que le falta al otro en el caso de validar datos en los cuales uno lo tenga y al otro le haga falta eso mismo.
  + Si queremos usar el wait/notifyAll, el wait SIEMPRE en la clase que no sea en el main y dentro de un bloque syncronize, sino frezeee de pantalla ya que con notifyAll esto no pasa porque rompe el wait y hace que se ejecuten todos los hilos al mismo ms
  + (Codigo de prueba en el de la carrera)
* IMPORTANTE:
* Nombres de paquetes: En minúsculas.
* Nombres de clases e interfaces: En CamelCase (la primera letra en mayúscula).
* Nombres de métodos y variables: En camelCase (la primera letra en minúscula).