Tarea Programada
Dayana Marín Mayorga
B64096
Programación paralela y concurrente

Ejercicio Collatz:

Este ejercicio se realizo un archivo llamado main.cpp
La idea general que es que se generen hijos mediante el método fork, utilizando memoria compartida para los procesos.

Métodos:
Método que verifica si un número es par o impar y hace la respectiva operación además calcula la cantidad de pasos que se hacen para llegar a uno con cada elemento.

```
//Metodo que verifica si el numero es par o impar y hace respectiva  operacion

int collatz(int num){
  int steps = 0;

  while (num != 1){
    if ((num % 2) == 0){
      num = num/2;
      steps++;
    } else {
      num = num * 3 + 1;
      steps++;
    }
  }

  return steps;
}
```

Método donde se crean los respectivos hijos, se crean 5 porque todo número potencia de 10 es divisible por 5.

```
  //Se crean los procesos hijos
  //5 procesos hijos
  padreID = getpid();
  for (int i = 1; i <= 5; i++){
    if (getpid() == padreID ){
      my_pid = i; // identificador de proceso interno
      my_shmid = my_pid+5000;
      shm_create( shm[i-1], my_shmid );
      hijoID = fork();
    } else {
      break;
    }
  }
```

Método que retorna el identificador de memoria compartida asociada a la llave.

```
void shm_get( int *& shm, int key ){
  int shmid;

  if ((shmid = shmget(key, 2*sizeof(int), 0666)) < 0) {
    perror("shmget");
    exit(1);
  }

  shm = (int *)shmat(shmid, NULL, 0);
  if((intptr_t) shm == -1){
    perror("Error en shmat");
    exit(1);
  }
}
```

Método del trabajo que hace cada proceso.
Se llama a shm_get para atar su dirección de memoria específica al proceso.
En el for se aplica collatz a cada elemento correspondiente, se hace repartición a todos los proceso.
De último se compara si los pasos que cada elemento genera es mayor del que se tiene registrado.

```
void process_work( int pid, int shmid, int l_limit, int u_limit ){
  int * my_shm;
  int steps;
  shm_get( my_shm, shmid );
  my_shm[0] = 0; // numero que genero la mayor cantidad de pasos
  my_shm[1] = 0; // cantidad de pasos maxima

  for(int i = l_limit+(pid-1); i <= u_limit; i += 5){
    steps = collatz(i);

    //Se comparara si la cantidad de pasos es mayor a la que ya se encuentra
//    y tambien se agrega el numero que genero esa cantidad de pasos
    if (steps > my_shm[1]){
      my_shm[0] = i;
      my_shm[1] = steps;
    }
  }
}
```

Se crea el área de memoria compartida de 8 bytes.
En el primer parámetro queda guardada la dirección de memoria del área que se creó.

```
void shm_create( int *& shm, int key ){
  int shmid;

  if ((shmid = shmget(key, 2*sizeof(int), IPC_CREAT | 0666)) < 0) {
    perror("shmget");
    exit(1);
  }

  shm = (int *)shmat(shmid, NULL, 0);
  if((intptr_t) shm == -1){
    perror("Error en shmat");
    exit(1);
  }
}
```

Segundo ejercicio:
Se crean 3 clases Ascensor, Persona y clase Mutex.
Se utilizó la plantilla dada por el profesor.

La idea general para la resolución de este ejercicio fue utilizar un vector de forma global llamado solicitudes donde se guardan todas las solicitudes de las personas.

Se genera un método para que en ese vector se guarden los diferentes elementos que se generan aleatoriamente de las personas.

```
vector< int > Persona::solicitud(char * rotulo){

vector< int > personas;
personas.push_back(this->idPersona);
personas.push_back(this->pisoDondeMeSubo);
personas.push_back(this->pisoDondeMeBajo);

return personas;
}
```