Se necesita estimar la probabilidad de que al lanzar *n* dados, su suma resulte ser *sum*. Para ayudar en la estimación, la función *favorables* simula lanzar *rep* veces los *n* dados, retornando la cantidad de casos favorables, es decir los casos en que la suma de los *n* dados fue *sum*. La probabilidad se estima como casos favorables / *rep*. Esta es la versión secuencial de *favorables*:

```
int favorables(int n, int sum, int rep) {
  int fav= 0;     // número de casos favorables
  for (int i= 0; i<rep; i++) {
    int s= 0;     // suma de los dados
    for (int j= 0; j<n; j++)
      s += rand_int(1,6); // entrega entero pseudo-aleatorio en [1,6]
    if (s==sum)   // caso favorable: los dados suman sum
      fav++;
  }
  return fav;
}
```

Programe la función *favorables_par* con el siguiente encabezado:

```
int favorables_par(int n, int sum, int rep, int p);
```

Esta función debe realizar la misma simulación que *favorables*, pero lo hace en paralelo usando *p* cores. Para lograrlo, Ud. debe invocar *p* veces *fork* para crear *p* procesos hijos. En cada hijo invoque la función *favorables* incluida en *test-fav.c* para simular *rep/p* lanzamientos de los *n* dados. Utilice un pipe por cada proceso hijo para que este le entregue al proceso padre la cantidad de casos favorables que obtuvo. El padre no realiza simulaciones de lanzamientos de los dados, solo suma y retorna la cantidad total de casos favorables. Puede usar la función *leer* incluida en *test-fav.c* para obtener las cantidades de casos favorables estimadas por los hijos.

Su función *favorables_par* (versión paralela) debe ser 1.5 veces más rápida que la función favorables (versión secuencial). Cuando pruebe su tarea con *make run* en su computador, asegúrese de que posea al menos 2 cores, que esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr el *speed up* solicitado.

Tenga presente que los números que entrega *rand_int* son pseudo aleatorios. Se generan a partir de una semilla y parecen tener una distribución aleatorio, pero si se usa la misma semilla en todos los procesos hijos, todos generarán la misma secuencia de números aleatorios y por lo tanto todos obtendrán la misma cantidad de casos favorables, invalidando el resultado final. Para hacer que los procesos hijos generen secuencias distintas de números pseudo aleatorios, invoque esta función justo ***antes*** de llamar a *fork*:

```
init_rand_seed();
```

La función *favorables* incluida en el test de prueba muestra en pantalla para cada proceso la semilla inicial usada y la cantidad de casos favorables que se obtuvo. Verifique que no son todos iguales. Si son todos iguales, Ud. no está invocando en el lugar correcto la función *init_rand_seed* y su tarea será rechazada.

Instrucciones

Baje *t7.zip* de U-cursos y descomprímalo. Ejecute el comando *make* sin parámetros. Le explicará cómo compilar y ejecutar, y los requisitos que debe cumplir su tarea para ser aprobada.

Entrega

Ud. debe entregar por U-cursos el archivo *fav.zip* que genera el comando *make zip*. **Recuerde descargar el archivo que subió, descomprimirlo e inspeccionar el contenido para verificar que son los archivos correctos.** Se descuenta medio punto por día de atraso, sin considerar recesos, sábados, domingos y festivos.