Se requiere calcular el producto de los enteros de un arreglo. El resultado puede ser muy grande y por lo tanto se usa el tipo *BigNum* que permite representar enteros positivos de tamaño arbitrario. Ya existe la función *seqArrayProd*, programada en *test-prod.c*, que calcula secuencialmente el producto usando divide y conquista. Su código es similar a este:

```
BigNum *seqArrayProd(int a[], int i, int j) {
  if (i==j) {
    return smallNum(a[i]); // Convierte un entero de C a BigNum
  }
  else {
    int h= (i+j)/2;
    BigNum *left= seqArrayProd(a, i, h);
    BigNum *right= seqArrayProd(a, h+1, j);
    // Multiplicacion de BigNum's
    BigNum *prod= bigMul(left, right);
    freeBigNum(left);   // Hay que liberar la memoria
    freeBigNum(right); // ocupada por los BigNum's
    return prod;
  }
}
```

Ud. debe programar en el archivo *prod.c* la función:

*BigNum *parArrayProd(int a[ ], int i, int j, int p);*

Esta función debe calcular el producto de los enteros *a[i]*, *a[i+1]*, …, *a[j]* usando divide y conquista, en paralelo con *p* procesos pesados. Deberá crear los procesos pesados con *fork*. Cuando *p* sea 1, Ud. debe invocar la función *seqArrayProd* que hace el cálculo en un solo proceso, secuencialmente. Lea las observaciones en la plantilla *prod.c.plantilla*, le ayudará a resolver este problema.

Ayuda: El hijo necesitará enviar su resultado en formato binario al padre a través de un pipe. Si en el proceso hijo, *b* es de tipo *BigNum**, envíelo a través del pipe *fds[1]* con:

```
write(fds[1], &b->n, sizeof(int));
write(fds[1], b->bits, b->n*sizeof(BigInt_t));
```

Y recíbalo del pipe *fds[0],* en el padre con:

```
BigNum *b= malloc(sizeof(BigNum));
leer(fds[0], &b->n, sizeof(int));
b->bits= malloc(b->n*sizeof(BigInt_t));
leer(fds[0], b->bits, b->n*sizeof(BigInt_t));
```

Instrucciones

Baje *t7.zip* de U-cursos y descomprímalo. El directorio *T7* contiene los archivos *test-prod.c*, *Makefile*, *prod.h* (con los encabezados requeridos) y otros archivos. Ud. debe crear el archivo *prod.c* y programar ahí la función *parArrayProd*. Ud. debe usar *fork* para resolver el problema. No puede usar threads.

Pruebe su tarea bajo Debian 10 de 64 bits. Ejecute el comando *make* sin parámetros. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run-O* debe felicitarlo por aprobar este modo de ejecución. El *speed up* reportado debe ser de al menos 1.2.
- *make run-g* debe felicitarlo. Además se verificará que al invocar *parArrayProd* con *p*=4, Ud. invoque 4 veces *seqArrayProd*.
- *make run-valgrind* debe felicitarlo, reportar 0 errores y solo debe reportar memory leaks del tipo *still reachable*. No se aceptara otro tipo de memory leaks.

Cuando pruebe su tarea con *make run-O* en su notebook 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.

Invoque el comando *make zip* para ejecutar todos los tests y generar un archivo *prod.zip* que contiene *prod.c*, con su solución, y *resultados.txt*, con la salida de *make run-O*, *make run-g* y make *run-valgrind*.

Opcionalmente ejecute el comando *make run-duende* y trate de explicar el comportamiento extraño del programa *duende.c* cuando se redirige su salida estándar a un archivo. No requiere hacer ninguna entrega con respecto al duende.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *prod.zip* generado por *make zip*. Verifique que adentro están *prod.c* y *resultados.txt*. Se descontará medio punto por día de atraso. No se consideran los días de vacaciones, sábado, domingo o festivos.