Bil 461 Hw-1

Fatih Furkan HAS
141101024

1-) Important behaviors'da denildiğine göre işlemler bitmeden diğerine geçilmeyecek. İşlemler bir önceki işlemin bitmesini bekleyecekler. IO işlemi de zaten çalıştırdığımız processlerde bulunmadığı için, CPU verimliliği %100 olacaktır. -c -p parametreleri ile çalıştırdığımızda da bu sonuçları görebiliyoruz.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 5:100,5:100 -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2     RUN:cpu      READY          1
  3     RUN:cpu      READY          1
  4     RUN:cpu      READY          1
  5     RUN:cpu      READY          1
  6        DONE    RUN:cpu          1
  7        DONE    RUN:cpu          1
  8        DONE    RUN:cpu          1
  9        DONE    RUN:cpu          1
 10        DONE    RUN:cpu          1

Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy  0 (0.00%)
```

2-) İlk işlem için CPU işlemleri yapılmıştır, bu işlem bittikten sonra ikinci işleme geçilmiştir ve ikinci işlem de bir IO işlemi olduğu için waiting durumuna geçmiştir, daha sonra 4 çevrim boyunca bu işlemi beklemiştir bir sonraki çevrimde de işlem tamamlanmıştır. İşlemlerin 10 zaman alması gerekiyordu. -c ve -p parametreleri ile çalıştırınca da bu durumu görebildik.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 4:100,1:0. -c -p
Time     PID: 0     PID: 1        CPU        IOs
  1     RUN:cpu      READY          1
  2     RUN:cpu      READY          1
  3     RUN:cpu      READY          1
  4     RUN:cpu      READY          1
  5        DONE     RUN:io          1
  6        DONE    WAITING                    1
  7        DONE    WAITING                    1
  8        DONE    WAITING                    1
  9        DONE    WAITING                    1
 10*       DONE       DONE

Stats: Total Time 10
Stats: CPU Busy 5 (50.00%)
Stats: IO Busy  4 (40.00%)
```

3-) Burada ise daha önce IO işleminin çalıştırılmasıyla CPU'nun boşuna beklemesinin önüne geçilmiştir. IO işlemleri yapılırken ikinci işlem CPU işlemlerini yapmıştır ve daha kısa sürede iki işlem de tamamlanmıştır. CPU verimliliği de artırılmıştır bu sayede.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 1:0,4:100. -c -p
Time     PID: 0     PID: 1        CPU        IOs
  1      RUN:io      READY          1
  2     WAITING    RUN:cpu          1          1
  3     WAITING    RUN:cpu          1          1
  4     WAITING    RUN:cpu          1          1
  5     WAITING    RUN:cpu          1          1
  6*       DONE       DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
```

4-) Burada SWITCH_ON_END parametresi ile IO işlemi bitmeden diğer işleme geçmesi engellenmiştir bu sebeple ikinci işlem CPU işlemi olmasına rağmen birinci işlemin IO işlemlerini tamamlanmasını beklemek zorunda kalmıştır. Programın çalışması daha uzun sürmüştür ve CPU verimliliği düşmüştür.

5-) 3. soru ile aynı çıktıyı ve çalışma zamanını elde ettik.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO -p
Time     PID: 0     PID: 1        CPU        IOs
  1      RUN:io      READY          1
  2     WAITING    RUN:cpu          1          1
  3     WAITING    RUN:cpu          1          1
  4     WAITING    RUN:cpu          1          1
  5     WAITING    RUN:cpu          1          1
  6*       DONE       DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
  1      RUN:io      READY          1
  2     WAITING      READY                 1
  3     WAITING      READY                 1
  4     WAITING      READY                 1
  5     WAITING      READY                 1
  6*       DONE    RUN:cpu          1
  7        DONE    RUN:cpu          1
  8        DONE    RUN:cpu          1
  9        DONE    RUN:cpu          1

Stats: Total Time 9
Stats: CPU Busy 5 (55.56%)
Stats: IO Busy  4 (44.44%)
```

6-) IO işlemlerini gereksiz yere CPU işlemleri sırasında bekletmiştir bu yüzden programın çalışması daha uzun sürmüştür ve verimlilik düşmüştür. IO işlemlerini CPU işlemleri sırasında yaparak çalışma zamanını kısaltabilirdi.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p
Time      PID: 0      PID: 1      PID: 2      PID: 3         CPU         IOs
  1       RUN:io       READY       READY       READY           1
  2      WAITING     RUN:cpu       READY       READY           1           1
  3      WAITING     RUN:cpu       READY       READY           1           1
  4      WAITING     RUN:cpu       READY       READY           1           1
  5      WAITING     RUN:cpu       READY       READY           1           1
  6*       READY     RUN:cpu       READY       READY           1
  7        READY        DONE     RUN:cpu       READY           1
  8        READY        DONE     RUN:cpu       READY           1
  9        READY        DONE     RUN:cpu       READY           1
 10        READY        DONE     RUN:cpu       READY           1
 11        READY        DONE     RUN:cpu       READY           1
 12        READY        DONE        DONE     RUN:cpu           1
 13        READY        DONE        DONE     RUN:cpu           1
 14        READY        DONE        DONE     RUN:cpu           1
 15        READY        DONE        DONE     RUN:cpu           1
 16        READY        DONE        DONE     RUN:cpu           1
 17       RUN:io        DONE        DONE        DONE           1
 18      WAITING        DONE        DONE        DONE                       1
 19      WAITING        DONE        DONE        DONE                       1
 20      WAITING        DONE        DONE        DONE                       1
 21      WAITING        DONE        DONE        DONE                       1
 22*      RUN:io        DONE        DONE        DONE           1
 23      WAITING        DONE        DONE        DONE                       1
 24      WAITING        DONE        DONE        DONE                       1
 25      WAITING        DONE        DONE        DONE                       1
 26      WAITING        DONE        DONE        DONE                       1
 27*        DONE        DONE        DONE        DONE

Stats: Total Time 27
Stats: CPU Busy 18 (66.67%)
Stats: IO Busy  12 (44.44%)
```

7-) Burada bir önceki işlemde olduğu gibi IO işlemlerini bekletmedik ve CPU işlemlerini yaparken IO işlemlerini çalıştırarak verimliliği artırdık ve programın çalışma zamanını kısalttık.

```
toor@001:~/Desktop/461Hw1/HW-CPU-Intro$ ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_IMMEDIATE -c -p
Time      PID: 0      PID: 1      PID: 2      PID: 3         CPU         IOs
  1       RUN:io       READY       READY       READY           1
  2      WAITING     RUN:cpu       READY       READY           1           1
  3      WAITING     RUN:cpu       READY       READY           1           1
  4      WAITING     RUN:cpu       READY       READY           1           1
  5      WAITING     RUN:cpu       READY       READY           1           1
  6*      RUN:io       READY       READY       READY           1
  7      WAITING     RUN:cpu       READY       READY           1           1
  8      WAITING        DONE     RUN:cpu       READY           1           1
  9      WAITING        DONE     RUN:cpu       READY           1           1
 10      WAITING        DONE     RUN:cpu       READY           1           1
 11*      RUN:io        DONE       READY       READY           1
 12      WAITING        DONE     RUN:cpu       READY           1           1
 13      WAITING        DONE     RUN:cpu       READY           1           1
 14      WAITING        DONE        DONE     RUN:cpu           1           1
 15      WAITING        DONE        DONE     RUN:cpu           1           1
 16*        DONE        DONE        DONE     RUN:cpu           1
 17         DONE        DONE        DONE     RUN:cpu           1
 18         DONE        DONE        DONE     RUN:cpu           1

Stats: Total Time 18
Stats: CPU Busy 18 (100.00%)
Stats: IO Busy  12 (66.67%)
```

Part 2:

1-) Oluşturduğumuz counter değişkeni her iki process için de farklı değişkenler olarak kullanılmıştır. İkisi de bu değişkeni 5'er kere arttırdığı halde değişken toplam 10 artmamıştır. Sadece 5 artmıştır. (kaynak kodu soru1.c)

2-) Child proccess'de tek basamaklı, main proccess'de ise iki basamaklı sayıları bastırdığımızda ilk başta main proccess çalıştı ve iki basamaklı sayıları ekrana bastırdı ancak bir süre sonra child proccess çalışarak araya girdi ve artan sırayla sayıları yazdırma işimizi böldü. Bu işlemi yapmak için bir wait() kullanmamız gerekmektedir. (kaynak kodu soru2.c)

3-) Bir fork() oluşturup child proccess'de execvp methodunu "echo" komutunu çalıştırmak için kullandım. "echo" komutu ile terminale "Hello FFH" yazdırmak istedim. Child proccessde bu işi yaptım execvp methodu parametre olarak bir komut ve komut için parametre arrayi almakta ancak child proccessin işlemi bitmediği için çalıştırdığımız program tam olarak sonlanmamakta. Exec komutunun farklı varyantları olmasının sebebi ise farklı parametreler vererek işlemleri çalıştırmak. Örneğin execve methodu environment variable'ları kullanmak için vardır. (kaynak kodu soru3.c)

4-) Yazdığım kodda child calisti, parent ise child'in bitmesine bekledi ve child bitti şeklinde ekrana mesaj bastı. Bunu yapmak için parent içinde wait() fonksiyonunu kullandım. Waitpid() ise spesifik bir çocuğu beklemek için kullanılmaktadır. (kaynak kodu soru4.c)

5-) Child proccess'de fclose(stdout) ile STDOUT u kapattık ve child proccess'de printf() ile ekrana bir şeyler yazdırmaya çalıştığımızda yazdıramadık. Ancak parentta yazdırmaya çalıştığımızda yazdırabildik. Bunun sebebi child'in ayri bi işlem olması. (kaynak kodu soru5.c)

6-) (kaynak kodu soru6.c)