Поправка за дисциплина “Операционни системи” (СИ), СУ, ФМИ, 25.08.2019 г.

Задача 1, СИ, 25.08

Множество паралелно работещи копия на процеса `P` изпълняват поредица от две инструкции:

```
process P
  p_1
  p_2
```

Осигурете чрез семафори синхронизация на работещите копия, така че:

Инструкцията `p_2` на всяко от работещите копия да се изпълни след като инструкция `p_1` е завършила изпълнението си в поне 3 работещи копия.

Упътване: Освен семафори, ползвайте и брояч.

Задача 2, СИ, 01.09

Опишете разликата между синхронни и асинхронни входно-изходни операции.

Дайте примери за програми, при които се налага използването на асинхронен вход-изход.

Примерни решения

Задача 1, СИ

За исканите в условието синхронизации използваме брояч `cnt` и два семафора – `m1` и `m2`, инициализираме ги така:

```
semaphore m1, m2
m1.init(1)
m2.init(0)
int cnt=0
```

Добавяме в кода на процеса `P` синхронизиращи инструкции:

```
process P
  p_1
  m1.wait()
   cnt=cnt+1
   if cnt=3 m2.signal()
  m1.signal()
  m2.wait()
  m2.signal()
  p_2
```

Семафорът `m1` ползваме като мутекс, който защитава брояча.

Стойността на `cnt` е равна на броя копия на процеса `P`, които са изпълнили своята първа инструкция.

Семафорът `m2` блокира изпълнението на инструкция `p_2`.

Когато третото копие на процеса `P` изпълни `p_1`, към семафора `m2` се подава сигнал, който го деблокира и позволява на всички копия да изпълнят втората си инструкция.

Задача 2, СИ

При синхронна входно-изходна операция системното извикване може да доведе до приспиване (блокиране) на потребителския процес, поръчал операцията.

Същевременно, при нормално завършване, потребителският процес разчита на коректно комлектоване на операцията – четене/запис на всички предоставени/поръчани данни във/от входно-изходния канал, или цялостно изпълнение на друг вид операция (примерно, изграждане на TCP връзка).

При асинхронна входно-изходна операция системното извикване не приспива (не блокира) потребителския процес, поръчал операцията.

Същевременно, при невъзможност да се комплектова операцията, ядрото връща управлението на процеса със специфичен код на грешка и друга информация, която служи за определяне на степента на завършеност на операцията.

Потребителският процес трябва да анализира ситуацията и при нужда да направи ново системно повикване по-късно, с цел да довърши операцията.

Използването на асинхронни операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран в случай на липса на входни данни, препълване на буфер за изходни данни или друга ситуация, водеща до блокиране.

Типични примери:

(1) Когато ползваме WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала. Браусерът проверява чрез асинхронни опити за четене по кой от каналите постъпва информация и реагира адекватно.

(2) Сървер в интернет може да обслужва много на брой клиентски програми, като поддържа отворени TCP връзки към всяка от тях. За да обслужва паралелно клиентите, сървърът трябва да ползва асинхронни операции, за да следи по кои връзки протича информация и кои са пасивни.

Когато програмата ползва асинхронни операции и никой от входно-изходните канали не е готов за обмен на данни, тя има нужда от специален механизъм за предоствяне на изчислителния ресурс на останалите процеси. Обикновено в такива случаи програмата се приспива сама за кратък период от време (в UNIX това става с извикване на sleep(), usleep() или nanosleep()).