Systemy Operacyjne 2017/18

Zadania - Zestaw 7

IPC - pamieć wspólna, semafory

Przydatne funkcje

System V:

```
<sys/shm.h> <sys/ipc.h> - shmget, shmclt, shmat, shmdt
```

POSIX:

```
<sys/mman.h> - shm_open, shm_close, shm_unlink, mmap, munmap
```

Problem synchronizacyjny śpiącego golibrody z kolejką

W ramach ćwiczenia 7 należy zaimplementować prawidłowe rozwiązanie problemu śpiącego golibrody z kolejką. Golibroda prowadzi zakład fryzjerski z jednym krzesłem w gabinecie (na którym strzyże klientów) i `N` krzesłami w poczekalni. Zakład działa według następującego schematu:

- **Golibroda**
  1. Golibroda sprawdza, czy w poczekalni oczekują klienci.
  2. Jeśli w poczekalni nie ma klientów, golibroda zasypia. W przeciwnym razie golibroda zaprasza do strzyżenia klienta, który czekał najdłużej.
  3. Gdy golibroda skończy strzyżenie, klient opuszcza zakład. Golibroda ponownie sprawdza poczekalnię.
- **Klient**
  1. Po przyjściu do zakładu klient sprawdza co robi golibroda.
  2. Jeśli golibroda śpi, klient budzi go, siada na krześle w gabinecie i jest strzyżony.
  3. Jeśli golibroda strzyże innego klienta, nowy klient sprawdza, czy w poczekalni jest wolne krzesło. Jeśli tak, to klient siada w poczekalni. W przeciwnym razie klient opuszcza zakład.

Program golibrody wypisuje na ekranie komunikaty o następujących zdarzeniach:

1. zaśnięcie golibrody,
2. obudzenie golibrody,
3. zaproszenie klienta z poczekalni do strzyżenia (w komunikacie tym podawany identyfikator strzyżonego klienta),
4. rozpoczęcie strzyżenia (z identyfikatorem strzyżonego klienta),
5. zakończenie strzyżenia (z identyfikatorem strzyżonego klienta).

Każdy klient wypisuje na ekranie komunikaty o następujących zdarzeniach:

1. obudzenie golibrody,
2. zajęcie miejsca na krześle do strzyżenia,
3. opuszczenie zakładu po zakończeniu strzyżenia,
4. zajęcie miejsca w poczekalni,
5. opuszczenie zakładu z powodu braku wolnych miejsc w poczekalni.

Każdy komunikat golibrody lub klienta powinien zawierać znacznik czasowy z dokładnością do mikrosekund (można tu wykorzystać funkcję `clock_gettime` z flagą `CLOCK_MONOTONIC`). Każdy komunikat klienta powinien ponadto zawierać jego identyfikator.

Zakładamy, że golibroda oraz każdy klient to osobne procesy. Klienci są tworzeni przez jeden proces macierzysty (funkcją `fork`). Identyfikatorem klienta jest jego `PID`. Proces tworzący klientów przyjmuje dwa argumenty: liczbę klientów do stworzenia oraz liczbę strzyżeń `S`. Każdy klient odwiedza w pętli zakład fryzjerski. Gdy klient zostanie ostrzyżony `S` razy, proces klienta kończy pracę. Proces macierzysty kończy pracę po zakończeniu wszystkich procesów klientów (można wówczas uruchomić w konsoli kolejną partię klientów). Proces golibrody pracuje do momentu otrzymania sygnału SIGTERM. Liczba krzeseł w poczekalni jest podawana w argumencie wywołania golibrody.

Synchronizacja procesów musi wykluczać zakleszczenia i gwarantować sekwencję zdarzeń zgodną ze schematem działania zakładu. Niedopuszczalne jest na przykład:

- budzenie golibrody, który nie śpi,
- zasypianie golibrody gdy w poczekalni czekają klienci,
- zajmowanie miejsca w poczekalni po obudzeniu golibrody (klient powinien od razu siąść na krześle do strzyżenia),
- rozpoczęcie strzyżenia klienta zanim klient zajmie miejsce na krześle do strzyżenia,
- opuszczenie zakładu (przez strzyżonego klienta) przed zakończeniem strzyżenia,
- itd.

Niedopuszczalne jest również zasypianie procesów klientów lub golibrody funkcjami z rodziny `sleep`.

Szeregowanie klientów do strzyżenia należy zaimplementować w postaci kolejki FIFO w pamięci wspólnej. W pamięci wspólnej można również przechowywać inne niezbędne zmienne (np. zmienną zliczającą klientów oczekujących w poczekalni, flagę wskazującą czy golibroda śpi, etc.). Odpowiednie zasoby do synchronizacji i komunikacji powinny być tworzone przez proces golibrody. Proces ten jest również odpowiedzialny za usunięcie tych zasobów z systemu (przed zakończeniem pracy).

Zadanie 1

Zaimplementuj opisany powyżej problem synchronizacyjny wykorzystując semafory i pamięć wspólną z IPC Systemu V.

Zadanie 2.

Zaimplementuj opisany powyżej problem synchronizacyjny wykorzystując semafory i pamięć wspólną z IPC POSIX.