# Systemy Operacyjne – laboratorium

*Ćwiczenie 3 – Semafory*

## 1. Treść zadania

Celem ćwiczenia jest napisanie programu w języku C w środowisku systemu Linux realizującego organizację bufora komunikacyjnego. Bufor komunikacyjny jest strukturą danych mieszczącą maksymalnie M elementów jednakowego typu. W zaproponowanej implementacji należy zapewnić synchronizację:

- nie dopuścić do czytania z pustego bufora;
- nie dopuścić do zapisu do pełnego bufora;
- zadbać o „nie przeszkadzanie” sobie procesów korzystających z bufora.

## 2. Przyjęte założenia

- długość kolejki – 100
- typy procesów korzystających z bufora:
  - producent (dodaje wartości do bufora)
  - konsument (pobiera wartości z bufora)
- typ przechowywanych danych - int

## 3. Proponowane rozwiązanie

Aby zapewnić odpowiednią synchronizację (opisaną w punkcie pierwszym), oprócz alokowania pamięci na bufor, należy stworzyć w systemie trzy semafory:

- **EMPTY** - przechowuje ilość pustych miejsc w kolejce (zapobiega zapisywaniu do pełnej kolejki)
- **FULL** - przechowuje ilość danych w kolejce (zapobiega czytaniu z pustego bufora)
- **MUTEX** – semafor binarny blokujący dostępu do bufora dla więcej niż jednego procesu

Do stworzenia i obsługi ww. semaforów oraz pamięci współdzielonej zostały wykorzystane wywołania systemowe systemu Linux:

- **shmget()** - alokacja pamieci współdzielonej;
- **semget()** - tworzenie semaforów;
- **semctl()** - ustawianie początkowych wartości semaforów;
- **shmctl()** - operacje na pamięci (dealokacja);
- **semop()** - podnoszenie/opuszczanie semaforów ;
- **shmat()** - dołączanie segmentu pamięci współdzielonej do przestrzeni adresowej procesu wywołującego;
- **shmdt()** - odłączanie segmentu pamięci współdzielonej od przestrzeni adresowej procesu wywołującego.

## 4. Testowanie

W celu implementacji i przetestowania omawianego bufora, stworzyłem 3 pliki wykonywalne:

- czytelnik – proces pobierający wartości z bufora;
- pisarz – proces dodający wartości do bufora;
- nadzorca – proces tworzący bufor komunikacyjny – alokujący pamięć, tworzący semafory, a następnie wywołujący losowo ww. procesy za pomocą funkcji **fork()** oraz **exec()**. Ilość wywoływanych procesów jest przekazywana w argumencie podczas uruchamiania testów.

*Autor: Maciej Suchecki – grupa 3I4*