5. Semafory

Systemy Operacyjne

Marcin Klimek

16 stycznia 2024

Politechnika Krakowska
Wydział Inżynierii
Elektrycznej i Komputerowej

Spis treści

1 Wstęp

Celem laboratorium było zapoznanie się z semaforami. Semafory są mechanizmem synchronizacji procesów, który pozwala na kontrolowanie dostępu do zasobów współdzielonych przez wiele procesów. W ramach laboratirum należało napisać programy, demonstrujące działania semafor, oraz sposoby w których może dojść do zakleszczenia.

2 Programy

2.1 Blokada Semafora

Program 1 demonstruje zachowanie semafora w przypadku, gdy jego wartość jest równa 0. W takiej sytuacji proces, który próbuje zająć semafor zostaje zablokowany do momentu, gdy semafor zostanie zwolniony przez inny proces.

```c
#include <semaphore.h>
#include <stdio.h>

int main() {
    sem_t Sem;
    int wartosc;
    int i;

    wartosc = 10;

    sem_init(&Sem, 0, wartosc);

    sem_getvalue(&Sem, &wartosc);
    printf("\nPoczątkowa wartość semafora 'Sem': %d\n", wartosc);

    for (i = wartosc + 5; i > 0; i--) {
        sem_wait(&Sem);
        sem_getvalue(&Sem, &wartosc);
        printf("Wartość semafora 'Sem' po opuszczeniu: %d\n", wartosc);
    }

    sem_destroy(&Sem);
    printf("Semafor 'Sem' został zniszczony.\n");

    return 0;
}  // main
```

Wyjście programu:

```
Początkowa wartość semafora 'Sem': 10
Wartość semafora 'Sem' po opuszczeniu: 9
Wartość semafora 'Sem' po opuszczeniu: 8
Wartość semafora 'Sem' po opuszczeniu: 7
Wartość semafora 'Sem' po opuszczeniu: 6
Wartość semafora 'Sem' po opuszczeniu: 5
Wartość semafora 'Sem' po opuszczeniu: 4
Wartość semafora 'Sem' po opuszczeniu: 3
Wartość semafora 'Sem' po opuszczeniu: 2
```

```
Wartość semafora 'Sem' po opuszczeniu: 1
Wartość semafora 'Sem' po opuszczeniu: 0
```

Jak widać na wyjściu, program nigdy nie dochodzi do linijki `printf("Semafor 'Sem' został zniszczony.\n");`, ponieważ proces nie może zająć semafora, gdyż jego wartość jest równa 0. Przez to, proces będzie bez końca próbował zająć semafor. Jeśli w pętli `for` za wartość początkową przyjmiemy wartość nie większą niż wartość semafora (1)0, program wykona się w pełni i zakończy się bez problemów.

2.2 Inkrementacja Semafora

Program 2 jest bardzo podobny do programu 1, jednak zamiast funkcji `sem_wait` używa funkcji `sem_post`, która zwiększa wartość semafora.

```c
#include <semaphore.h>
#include <stdio.h>

int main() {
    sem_t Sem;
    int wartosc;
    int i;

    wartosc = 10;

    sem_init(&Sem, 0, wartosc);

    sem_getvalue(&Sem, &wartosc);
    printf("\nPoczątkowa wartość semafora 'Sem': %d\n", wartosc);

    for (i = wartosc; i > 0; i--) {
        sem_post(&Sem);
        sem_getvalue(&Sem, &wartosc);
        printf("Wartość semafora 'Sem' po opuszczeniu: %d\n", wartosc);
    }

    sem_destroy(&Sem);
    printf("Semafor 'Sem' został zniszczony.\n");

    return 0;
}  // main
```

Wyjście programu:

```
Początkowa wartość semafora 'Sem': 10
Wartość semafora 'Sem' po opuszczeniu: 11
Wartość semafora 'Sem' po opuszczeniu: 12
Wartość semafora 'Sem' po opuszczeniu: 13
Wartość semafora 'Sem' po opuszczeniu: 14
Wartość semafora 'Sem' po opuszczeniu: 15
Wartość semafora 'Sem' po opuszczeniu: 16
Wartość semafora 'Sem' po opuszczeniu: 17
Wartość semafora 'Sem' po opuszczeniu: 18
Wartość semafora 'Sem' po opuszczeniu: 19
```

```
Wartość semafora 'Sem' po opuszczeniu: 20
Semafor 'Sem' został zniszczony.
```

Zgodnie z oczekiwaniami, program wykonuje się bez problemów i kończy się.

2.3 Nieblokujący sem_trywait

W programie 3 przedstawiona jest funkcja `sem_trywait`, która próbuje opuścić semafor, ale nie blokuje procesu, jeśli semafor jest już na wartości 0.

```c
#include <stdio.h>
#include <semaphore.h>

int main() {
    sem_t Sem; // Definicja semafora.
    int wartosc, result, i;

    wartosc = 10; // Ustawienie wartości początkowej semafora na 10.

    // Inicjalizacja semafora lokalnego dla procesu (0 oznacza semafor lokalny).
    sem_init(&Sem, 0, wartosc);

    // Pobranie i wyświetlenie bieżącej wartości semafora.
    sem_getvalue(&Sem, &wartosc);
    printf("\nPoczątkowa wartość semafora: %d\n", wartosc);

    // Iteracja przez wartości semafora od wartości początkowej do 0.
    for (i = wartosc; i >= 0; i--) {
        // Próba 'trywait' na semaforze - nieblokujące zmniejszenie wartości.
        result = sem_trywait(&Sem);

        // Pobranie i wyświetlenie aktualnej wartości semafora po 'trywait'.
        sem_getvalue(&Sem, &wartosc);

        if (result == 0) {
            // Sukces 'trywait' - semafor został zmniejszony.
            printf("Wartość semafora po 'sem_trywait': %d\n", wartosc);
        } else {
            // Niepowodzenie 'trywait' - semafor nie został zmniejszony, ponieważ jest już na 0.
            printf("Semafor nie został zmniejszony - jest już na wartości 0. Aktualna wartość: %d\n",
        }
    }

    // Niszczenie semafora i zwolnienie zasobów.
    sem_destroy(&Sem);
    printf("Semafor został zniszczony.\n");

    // Wyjście z programu.
    return 0;
} // main
```

Wyjście programu:

```
Początkowa wartość semafora: 10
Wartość semafora po 'sem_trywait': 9
Wartość semafora po 'sem_trywait': 8
Wartość semafora po 'sem_trywait': 7
Wartość semafora po 'sem_trywait': 6
Wartość semafora po 'sem_trywait': 5
Wartość semafora po 'sem_trywait': 4
Wartość semafora po 'sem_trywait': 3
Wartość semafora po 'sem_trywait': 2
Wartość semafora po 'sem_trywait': 1
Wartość semafora po 'sem_trywait': 0
Semafor nie został zmniejszony - jest już na wartości 0. Aktualna wartość: 0
Semafor został zniszczony.
```

Jak widać na wyjściu, funkcja `sem_trywait` nie zmniejsza wartości semafora, jeśli jest ona równa 0. Program również zamiast być zablokowanym, kontynuuje swoje działanie, zależnie od wartości `result` zwracanej przez funkcję `sem_trywait`.

2.4 Synchronizacja Wątków

Progam 4 demonstruje działanie semafor wykorzystując wątki.

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void *Down(void *arg);
void *Up(void *arg);

static sem_t Sem;

int main() {
    pthread_t Tid_Up, Tid_Down;

    sem_init(&Sem, 0, 5);

    pthread_create(&Tid_Down, NULL, Down, NULL);
    pthread_create(&Tid_Up, NULL, Up, NULL);

    pthread_join(Tid_Up, NULL);
    pthread_join(Tid_Down, NULL);

    sem_destroy(&Sem);

    exit(0);
}

void *Down(void *arg) {
    int wartosc;
    int i;
```

```
    int result;

    for (i = 1; i <= 10; i++) {
        sem_getvalue(&Sem, &wartosc);
        printf("\n[DOWN]: Sem=%d ", wartosc);

        result = sem_trywait(&Sem);

        if (result == 0) {
            printf(" - Wątek kontynuuje działanie.\n");
        } else {
            printf(" - Wątek zablokowany, wartość result: %d\n", result);
            sem_wait(&Sem);
        }

        sleep(2);
    }

    return NULL;
}

void *Up(void *arg) {
    int wartosc;
    int i;

    for (i = 1; i <= 10; i++) {
        sem_getvalue(&Sem, &wartosc);
        printf("\n[UP]: Sem=%d ", wartosc);

        sem_post(&Sem);

        sem_getvalue(&Sem, &wartosc);
        printf(" - Po operacji sem_post: Sem=%d\n", wartosc);

        sleep(4);
    }

    return NULL;
}
```

W kodzie programu 4 widać, że wątek `Down` próbuje opuścić semafor, a wątek `Up` próbuje go zwiększyć. Wątek `Down` próbuje opuścić semafor za pomocą funkcji `sem_trywait`, która nie blokuje wątku, jeśli semafor jest na wartości 0. W takiej sytuacji, wątek `Down` zostaje zablokowany za pomocą funkcji `sem_wait`. Wątek `Up` zwiększa wartość semafora za pomocą funkcji `sem_post`.

Wyjście programu:

```
[DOWN]: Sem=5  - Wątek kontynuuje działanie.
[UP]: Sem=5  - Po operacji sem_post: Sem=5
[DOWN]: Sem=5  - Wątek kontynuuje działanie.
[UP]: Sem=4  - Po operacji sem_post: Sem=5
[DOWN]: Sem=5  - Wątek kontynuuje działanie.
[DOWN]: Sem=4  - Wątek kontynuuje działanie.
```

```
[UP]: Sem=3  - Po operacji sem_post: Sem=4
[DOWN]: Sem=4  - Wątek kontynuuje działanie.
[DOWN]: Sem=3  - Wątek kontynuuje działanie.
[UP]: Sem=2  - Po operacji sem_post: Sem=3
[DOWN]: Sem=3  - Wątek kontynuuje działanie.
[DOWN]: Sem=2  - Wątek kontynuuje działanie.
[UP]: Sem=1  - Po operacji sem_post: Sem=2
[DOWN]: Sem=2  - Wątek kontynuuje działanie.
[DOWN]: Sem=1  - Wątek kontynuuje działanie.
[UP]: Sem=0  - Po operacji sem_post: Sem=1
[UP]: Sem=1  - Po operacji sem_post: Sem=2
[UP]: Sem=2  - Po operacji sem_post: Sem=3
[UP]: Sem=3  - Po operacji sem_post: Sem=4
[UP]: Sem=4  - Po operacji sem_post: Sem=5
```

2.5 Przypadek Zakleszczenia

Celem programu 5 jest demonstracja przypadku, w którym dochodzi do zakleszczenia.

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

static sem_t P;
static sem_t Q;
static sem_t R;

void* W1(void* arg) {
  while(1) {
    sem_wait(&P);
    sem_wait(&Q);
    sem_wait(&R);

    printf("W1\n");

    sem_post(&Q);
    sem_post(&P);
    sem_post(&R);
  }
}

void* W2(void* arg) {
  while(1) {
    sem_wait(&R);
    sem_wait(&P);
    sem_wait(&Q);

    printf("W2\n");

    sem_post(&Q);
```

```
    sem_post(&P);
    sem_post(&R);
  }
}

void* W3(void* arg) {
  while(1) {
    sem_wait(&Q);
    sem_wait(&R);
    sem_wait(&P);

    printf("W3\n");

    sem_post(&Q);
    sem_post(&P);
    sem_post(&R);
  }
}

void* print_semaphores(void* arg) {
  while(1) {
    int p, q;
    sem_getvalue(&P, &p);
    sem_getvalue(&Q, &q);
    printf("P: %d, Q: %d\n", p, q);
    sleep(1);
  }
}

int main() {
  pthread_t w1, w2, w3, print;

  sem_init(&P, 0, 1);
  sem_init(&Q, 0, 1);
  sem_init(&R, 0, 1);

  pthread_create(&w1, NULL, W1, NULL);
  pthread_create(&w2, NULL, W2, NULL);
  pthread_create(&w3, NULL, W3, NULL);

  pthread_join(w1, NULL);
  pthread_join(w2, NULL);
  pthread_join(w3, NULL);

  sem_destroy(&P);
  sem_destroy(&Q);
  sem_destroy(&R);

  return 0;
}
```

Wyjście programu:

```
W1
W1
W1
W1
W1
W1
W1
W1
```

W programie 5 mamy 3 wątki, które próbują opuścić semafory w różnej kolejności. Wątek `W1` próbuje opuścić semafor `P`, `Q` i `R`. Wątek `W2` próbuje opuścić semafor `R`, `P` i `Q`. Wątek `W3` próbuje opuścić semafor `Q`, `R` i `P`. Wątki wykonują się w nieskończoność, ponieważ każdy z nich próbuje opuścić semafor, który jest zajęty przez inny wątek. Aby rozwiązać problem, należy zmienić kolejność opuszczania semaforów w każdym z wątków.

3 Wnioski

Semafor jest mechanizmem synchronizacji procesów, który pozwala na kontrolowanie dostępu do zasobów współdzielonych przez wiele procesów. W ramach laboratorium udało się zapoznać z podstawowymi funkcjami semaforów, oraz sposobami ich użycia. Podczas laboratoriów również udało się zauważyć, że semafory mogą powodować zakleszczenie, jeśli nie są użyte poprawnie.