Paweł Świątkowski

**Dokumentacja Projekt 3 SOI**

1. Mamy 5 kolejek FIFO o pojemności 10 elementów. Producenci mogą zapisywać do dowolnej z tych kolejek (losują do której), konsumenci natomiast związanie są z jedną konkretną kolejką i tylko z niej wyjmują elementy. Mamy 5 typów konsumentów – po jednym na każdą kolejkę.
2. Tworzymy 5 buforów – kolejek FIFO – zrealizowane na tablicy, buforze cyklicznym. Każda z kolejek mieści 10 elementów.

**Implementacja struktury kolejek:**

#define BUFSIZE 9

struct Queue {

char buf[BUFSIZE];

int head, tail, length;

};

Head-numer indexu w tablicy buf początku kolejki

Tail-numer indexu w tablicy buf końca kolejki

Length-ilość elementów w kolejce

**Dodawanie elementu do kolejki:**

void putToBuf(struct Queue \*q, char val) {

q->buf[q->tail] = val;

q->tail = (q->tail+1)%BUFSIZE;

q->length++;

printf("Wsadzam %c, size: %d\n", val, q->length);

}

**Wyjmowanie elementu z kolejki:**

char getFromBuf(struct Queue \*q){

char ret = q->buf[q->head];

q->head = (q->head+1)%BUFSIZE;

q->length--;

printf("Wyjmuje %d\n", ret);

return ret;

}

1. Wewnątrz głównej funkcji programu – main() tworzymy zadaną ilość procesów producentów oraz 5 konsumentów. Realizuję to za pomocą funkcji fork():

pid[i]=fork();

if(pid[i]==0){ Producent(i); }

pid[i]=fork();

if(pid[i]==0){ Consumer(i); }

Następnie usypiam funkcję main na zadany czas (50 sekund). W tym czasie będzie można obserwować dane wypisywane przez procesy wstawiające i wyjmujące elementy z kolejek. Po tym czasie zabija stworzone procesy za pomocą funkcji:

kill(pid[i], SIGKILL);

1. Korzystam z pamięci współdzielonej między procesami. Znajduje się w niej 5 elementowa tablica kolejek:

int buffid = shmget(BUFFKEY, 5\*sizeof(struct Queue), IPC\_CREAT|0600);

struct Queue \*buffer = (struct Queue\*)shmat(buffid, NULL, 0);

1. Wykorzystywał będę semafory wbudwane. Udostępniają one nam funkcje:

int semget ( key\_t key, int ilosc\_semaforow, int semflags ); - uzyskanie dostępu do grupy semaforów

int semctl ( int id\_semafora, int numer\_semfora ,int polecenie, wartość\_ktora\_bedzemy\_inicjowac\_semafor ) – będziemy używać do inicjacji początkowej wartości semaforów (polecenie = SETVAL) oraz do sprawdzania wartości danego semafora (polecenie = GETVAL), wtedy nie podajemy czwartego argumentu

int semop ( int semid, stuct sembuf \*sops, unsigned nsops ) – niepodzielna operacja która wykonuje zwiększenie semafora o podaną wartość (również zmniejszenie, jeżeli podamy wartość ujemną)

Będę używał 17 semaforów.

- 5 semaforów empty – zliczające ilość pustych elementów w każdej z kolejek, każdy inicjowany wartością 10 (na początku każda kolejka jest pusta czyli ma 10 pustych elementów).

- 5 semaforów full – zliczające ilość pełnych elementów w każdej z kolejek, każdy inicjowany wartością 0 (na początku każda kolejka jest pusta czyli ma 0 pełnych elementów).

- 5 semaforów mutex – semafory binarne strzegące dostępu do każdej z kolejki kiedy chcemy na niej dokonać operacji dodania lub usunięcia elementu, inicjowane wartością 1.

- 1 semafor groupEmpty – semafor zliczający ilość pustych elementów we wszystkich kolejkach razem, inicjowany wartością ILOSC\_KOLEJEK\*BUF\_SIZE=50 (każda z kolejek na początku jest pusta, więc w sumie mamy 50 pustych elementów). Wykorzystywany on będzie aby wstrzymać konsumentów którzy będą czekać aż chociaż jedna z kolejek nie będzie pełna i wtedy będą wybierali która jest nie pełna i to właśnie do niej będą wstawiali.

- 1 semafor mutexProd – semafor binarny wykorzystywany do zablokowania producentów gdy jeden z nich wybiera sobie do której kolejki będzie wstawiał i jednocześnie po wybraniu nie pełnej kolejki do której będzie mógł wstawić będzie już rezerwował do niej dostęp aby wstawić, inicjowany wartością 1.

Na semaforach wykonywane są operacje UP oraz DOWN. Operacja UP zwiększa zadany semafor o podaną wartość, natomiast operacja DOWN próbuje zmniejszyć semafor o podaną wartość, gdy jest to niemożliwe usypia się do czasu gdy stanie się to możliwe i wykonuje tą operację.

Obydwie operacje opierają się na niepodzielnej operacji semop() – zaimplementowanej w systemie.

1. Proces Producenta – będzie używał pamięci współdzielonej (5 kolejek) oraz semaforów. Konsument wsadza jeden element do wylosowanej kolejki.

Proces na początku losuje sobie listę kolejek do których będzie chciał wstawić elementy (tak naprawdę to miesza tablicę z numerami wszystkich kolejek, symuluje to losowanie bez powtórzeń spośród 5 kolejek). Następnie wykonuje operację DOWN(groupEmpty), aby sprawdzić czy jest jakakolwiek pusta kolejka (jeśli nie to usypia w tym miejscu), następnie blokuje innych producentów (DOWN(mutexProd)) aby wybrać pierwszą wolną kolejkę spośród swoich wylosowanych. Blokuje innych aby mieć pewność że nikt wcześniej nie wsadzi elementu do tej kolejki którą sobie wylosuje. Następnie robi operacje DOWN(empty, wylosowanaKolejka) – czyli zmniejsza semafor empty tej kolejki do której będzie wsadzał element (takie logiczne wsadzenie elementu do kolejki – inni producencie będą już myśleli że do danej kolejki został wstawiony element, fizyczne wstawienie odbędzie się później). Po tym odblokowuje innych producentów – UP(mutexProd). Następnie wykonuje operację DOWN(mutex, wylosowanaKolejka) – aby zablokować dostęp do zadanej kolejki na czas gdy faktycznie będzie wstawiał do niej element. Po wstawieniu odblokowuje dostęp do tej kolejki czyli robi UP(mutex, wylosowanaKolejka). Następnie zwiększa semafor informujący o ilość pełnych elementów zadanej kolejce – UP(full, wylosowanaKolejka). Po tych operacjach zasypia na określoną ilość czasu – fukcja sleep(ilość\_sekund).

1. Proces Konsumenta będzie używał pamięci współdzielonej (5 kolejek) oraz semaforów. Konsument wyjmuje jeden element z kolejki takiego numeru jaki on sam ma.

Proces na początku wykonuje operacje DOWN(full, numerKolejki) aby czekać aż dana kolejka nie będzie pusta – czyli aż będzie mógł z niej wyjąć element. Po tym robi DOWN(mutex, numerKolejki) – blokuje dostęp do zadanej kolejki na czas gdy będzie z niej wyjmował element. Po wyjęciu elementu odblokowuje dostęp do tej kolejki wywołując UP(mutex, numerKolejki). Po tym wykonuje UP(empty, numerKolejki) informując że wyjął element z zadanej kolejki – zwiększa o 1 semafor empty. Na koniec proces konsumenta zostaje uśpiony na zadany czas – sleep(ilość\_sekund).