4 Komunikacja międzyprocesowa. Pamięć wspólna i kolejki komunikatów.

System wielozadaniowy oprócz mechanizmów synchronizacji potrzebuje również niezawodnych mechanizmów komunikacji między tymi zadaniami. Często działanie systemów polega na tym, że jedna grupa wątków zajmuje się pozyskiwaniem danych (np. z urządzenia zewnętrznego) i przekazywaniem ich do innej grupy wątków, które te dane analizują i przetwarzają. Następnie dane zazwyczaj przekazywane są dalej np. w celu ich prezentacji. Tak więc solidna droga komunikacji między grupami tych wątków jest niezbędna. Jak można zauważyć mechanizmy komunikacji są ściśle związane z synchronizacją. Jeden wątek pisze, drugi czyta w odpowiednim czasie.

Jednym z ważniejszych mechanizmów jest komunikacja poprzez pamięć wspólną. Jest to obszar pamięci, do której dostęp może mieć wiele wątków pisząc i czytając bezpośrednio w niej. W tym przypadku komunikacji niezbędne są mechanizmy ochrony sekcji krytycznej. Ogromną zaletą tego rozwiązania jest szybkość dostępu. W Xenomai obsługa pamięci współdzielonej wygląda następująco:

```
RT_HEAP heap;
char   *heap_buffer;
...
rt_heap_create(&heap, "heap", 1024, H_SHARED);
rt_heap_alloc(&heap, 0, TM_NONBLOCK, (void **)&heap_buffer);
...
memcpy(heap_buffer, buffer, BUFF_LEN);
...
rt_heap_delete(&heap);
```

`heap_buffer` jest to wskaźnik na początek obszaru pamięci wspólnej. Pisząc pod ten adres (lub kopiując) dane umieszczane są w tym obszarze wspólnym. Po informacje na temat konkretnych parametrów tych funkcji odsyłam do dokumentacji `(Memory heap services)`.

(a) Wykorzystaj napisany program z poprzednich zajęć (`ex3/b`) i zmodyfikuj go w taki sposób, żeby pisarz i czytelnik komunikowali się poprzez mechanizm pamięci wspólnej. Zadanie należy wykonać w dwóch plikach: ex4a_writer.c i ex4a_reader.c. ex4a_writer.c alokuje segment

pamięci wspólnej, tworzy semafory oraz wątek pisarza. ex4a_reader.c za pomocą funkcji `*_bind` ma pobrać wskaźniki na te utworzone obiekty oraz utworzyć wątek czytelnika. Wątki mogą dzielić wspólne obiekty, odbywa się to dzięki ich identyfikacji we wspólnej przestrzeni nazw. Tak więc program ex4a_reader.c chcąc korzystać z semafora utworzonego przez ex4a_writer.c musi to zrobić w przedstawiony niżej sposób:

```
RT_SEM empty, full;
rt_sem_bind(&empty, "empty", TM_NONBLOCK);
rt_sem_bind(&full, "full", TM_NONBLOCK);
```

Warunkiem jest wcześniejsze utworzenie tych obiektów przez inny wątek, oraz podanie poprawnych nazw. Należy również pamiętać o późniejszym wywołaniu funkcji `rt_sem_unbind` na tych obiektach. Po stronie czytelnika obsługa pamięci wspólnej wygląda następująco:

```
RT_HEAP heap;
char   *heap_buffer;
...
rt_heap_bind(&heap, "heap", TM_NONBLOCK);
rt_heap_alloc(&heap, 0, TM_NONBLOCK, (void **)&heap_buffer);
...
memcpy(buffer, heap_buffer, BUFF_LEN);
...
rt_heap_unbind(&heap);
```

(b) Równie ważnym mechanizmem jest komunikacja poprzez kolejki. Jest to liniowa struktura danych, do której informacje mogą być dopisywane, oraz z niej pobierane w ustalonej kolejności. Bufor kolejki ma określoną długość, natomiast pobrane dane są z kolejki usuwane.

Zadaniem jest napisanie programu do równoległego obliczenia przybliżonej wartości liczby π. Program ma składać się z jednego wątku głównego `server` i N wątków obliczeniowych `clients[N]`. Gdy wątek główny „obudzi" pozostałe wątki każdy z nich wykonuje własne obliczenia i po skończeniu odsyła wynik do wątku głównego, który po zebraniu wszystkich wyświetla rezultat całej operacji.
Przybliżenie liczby π można uzyskać w następujący sposób. Posiadając kwadrat o boku 1, oraz wpisane w niego koło o promieniu 0.5 trzeba wygenerować równomiernie rozłożony zestaw B

punktów wewnątrz tego kwadratu. Za pomocą A oznaczona jest liczba punktów wśród wylosowanych, które znajdują się wewnątrz koła $x^2 + y^2 <= r^2$. Za pomocą przedstawionego równania można otrzymać przybliżenie liczby π:

$$\frac{A}{B * r^2} \tag{1}$$

Przybliżenie rośnie wraz z liczbą wylosowanych punktów. Najważniejszą zaletą tego sposobu jest to, że obliczanie może przebiegać równolegle, na wielu wątkach niezależnych od siebie. Wystarczy, że każdy wątek wylosuje własny zestaw punktów i sprawdzi ich obecność w kole. Wtedy A będzie oznaczać sumę tych liczb.
Poniżej przedstawiony jest ideowy schemat działania wątku głównego i obliczeniowego:

```
SERWER
1. wyślij do wszystkich wątków liczbę B/N (broadcast)
2. dla wszystkich wątków:
3.   oczekuj (nieskończenie) na wynik
4.   zsumuj z poprzednimi
5. wykonaj równanie i wyświetl wyliczoną wartość PI

KLIENT
1. oczekuj (nieskończenie) na komunikat od serwera
2. dla otrzymanej liczby iteracji:
3.   wylosuj punkt (x,y) wewnątrz kwadratu (drand48)
4.   sprawdź czy zawiera się w kole i zwiększ licznik
5. wyślij do serwera paczkę z nazwą wątku i uzyskaną liczbą
```

Zapoznaj się z funkcjami obsługi kolejek, zwłaszcza `rt_queue_create`, `rt_queue_read`, `rt_queue_write` (`(Message queue services)`). Przy tworzeniu kolejki należy pamiętać aby limit maksymalnej liczby wiadomości w kolejce ustawić adekwatnie do liczby wątków.

```
rt_queue_create(&queue, "queue", 255, N, 0);
```

Aby wysłać wiadomość do wszystkich oczekujących wątków należy użyć flagi `Q_BROADCAST`.

```
rt_queue_write(&queue, &buf, sizeof(buf), Q_BROADCAST);
```

Należy również pamiętać o odpowiedniej ochronie (np. muteks) zapisu do kolejki przy wysyłaniu wyników. Program dla 10 wątków i 10 000 000 punktów wygląda następująco:

```
client1 found 785651/1000000 points
client8 found 784955/1000000 points
client7 found 785121/1000000 points
client3 found 785779/1000000 points
client4 found 785510/1000000 points
client6 found 785588/1000000 points
client9 found 785041/1000000 points
client0 found 785650/1000000 points
client2 found 785670/1000000 points
client5 found 784952/1000000 points
PI = 3.141566800
```