UXP1.A Zadanie 1

Projekt Wstępny

# Zespół projektowy

* Adrian Wiśniewski
* Paweł Witkowski

# Podstawowa struktura rozwiązania

Węzły systemu zostaną zaimplementowane jako oddzielne procesy komunikujące się ze sobą za pomocą potoków systemu Unix. Wśród nich wyróżniony będzie jeden węzeł pełniący rolę magazynu krotek. Węzeł ten będzie posiadał jeden potok wejściowy na którym nasłuchiwać będzie żądań wysyłanych przez pozostałe procesy i po jednym potoku wyjściowym dla każdego z tych procesów którym będzie wysyłał wyniki operacji.

Potok Żądań

Potok Odpowiedzi

Potok Odpowiedzi

## Proces magazynujący

Proces ten jest rodzicem procesów roboczych, ponieważ istnieje przez całe życie systemu i musi przekazywać deskryptory potoków powstającym procesom roboczym. Działa w pętli pobierając kolejne żądania z wejściowego potoku żądań, przy czym będzie przetwarzać tylko jedno żądanie w czasie, co pozwala na osiągnięcie atomowości wykonywanych operacji. Po przetworzeniu żądania wysyła jago wynik do procesu roboczego

## Proces roboczy

Wykonuje logikę aplikacyjną, komunikując się z procesem magazynującym za pomocą wywołań dostarczonych funkcji bibliotecznych.

## Biblioteka funkcji Linda

Udostępnia operacje języka Linda procesowi roboczemu, ukrywając przed nim szczegóły techniczne.

## 

# Podstawowe komunikaty

## Ogólny schemat komunikatu

Każdy komunikat składa się z nagłówka i danych. Nagłówek zawiera dwa pola typu całkowitoliczbowego: rozmiar komunikatu i typ komunikatu. Rozmiar pozwala wczytać komunikat w całości do pamięci, a typ określić sposób przetwarzania i występujące w nim pola danych.

|  |  |  |
| --- | --- | --- |
| Rozmiar|int | Typ|int | Dane |

## Operacje odczytu – input i read

### Komunikaty żądań

Komunikat wysyłany przez proces roboczy w celu odczytania krotki. Zawiera pid procesu nadającego, aby proces magazynujący wiedział do którego procesu należy odesłać odpowiedź. Ponadto zawiera zserializowaną strukturę query, będącą formatem wynikowej krotki.

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar|int | Typ|int | Pid|int | Zapytanie|Linda::Query |

### Przetwarzanie

W sytuacji, gdy odczyt jest możliwy, proces magazynujący natychmiast odsyła procesowi roboczemu wynik żądania. W przeciwnym wypadku żądanie to zostanie odłożone do kolejki fifo odczytów oczekujących. Oczekujący proces roboczy zawiesi się na operacji odczytu wyniku, aż do czasu jego nadejścia.

### Komunikaty odpowiedzi

Proces magazynujący odsyła żądaną krotkę w postaci zserializowanej.

|  |  |  |
| --- | --- | --- |
| Rozmiar|int | Typ|int | Krotka|Linda::Tuple |

## Operacja zapisu – output

### Komunikaty żądań

Komunikat wysyłany przez proces roboczy w celu zapisania krotki. pid procesu nadającego, aby proces magazynujący wiedział do którego procesu należy odesłać odpowiedź i samą krotkę w postaci zserializowanej.

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar|int | Typ|int | Pid|int | Krotka|Linda::Tuple |

### Przetwarzanie

Proces magazynujący od razu wysyła procesowi roboczemu wynik operacji, a następnie przeszukuje kolejkę odczytów oczekujących w poszukiwaniu żądań spełnionych przez zapisywaną krotkę. W razie znalezienia takich żądań proces ten odsyła wynik odczytu i usuwa żądania z kolejki odczytów oczekujących. Dopiero gdy w kolejce nie ma żądań oczekujących na daną krotkę, lub są ale tylko typu read, krotka zostaje zapisana do magazynu.

### Komunikaty odpowiedzi

Proces magazynujący odsyła komunikat o typie success w razie sukcesu.

|  |  |
| --- | --- |
| Rozmiar|int | Typ|int |

## Dodatkowe założenia

* Wielkość krotek i zapytań nie przekracza wartości PIPE\_BUF minus rozmiar nagłówka komunikatu. Jeżeli wielkość nie byłaby ograniczona, większe komunikaty należałoby podzielić na ramki, tak aby zostały one zapisane do potoków atomowo, a następnie po odbiorze składać ramki powrotem w kompletne komunikaty.

# Testowanie – struktura rozszerzona

Aby umożliwić dokładne i automatyczne testowanie rozwiązania zostanie utworzony proces testera, który przyjmuje polecenia wprowadzane z konsoli – w szczególności skrypty testujące przekierowane na wejście. Między procesami zostaną utworzone dodatkowe potoki służące do przekazywania rozkazów i ich wyników.

Potok Wyników

Potok Rozkazów

Potok Rozkazów

Potok Odpowiedzi

Potok Odpowiedzi

Potok Żądań

Potok Rozkazów

## Lista rozkazów

* create id – tworzy proces roboczy o podanym identyfikatorze
* kill id – niszczy proces o podanym identyfikatorze
* status – wyświetla zawartość magazynu krotek i listę odczytów oczekujących
* input id query – wysyła do procesu id rozkaz odczytania krotki zgodnej z query
* read id query – j.w. odczyt nie usuwający
* output id tuple – wysyła do procesu id rozkaz zapisania krotki tuple

## Proces testera

Odczytuje rozkazy z konsoli i wysyła je do procesu magazynującego. Ze względu na istnienie operacji blokujących wyniki nie muszą nadchodzić synchronicznie, dlatego każdy rozkaz i wynik jest oznaczony numerem identyfikacyjnym. Co jakiś czas sprawdza czy nadszedł wynik i wypisuje go na standardowe wyjście błędów jako log. W przypadku wykonania całego skryptu bezbłędnie wypisuje na standardowe wyjście komunikat „pass”, a w przypadku błędu wynik błędnej operacji i komunikat „fail”.

## Proces magazynujący

Nasłuchuje za pomocą funkcji *select* na deskryptorach potoków żądań procesów roboczych i rozkazów procesu testera. W przypadku pojawienia się komunikatu postępuję:

* Żądanie Linda procesu roboczego – jak w opisie rozwiązania podstawowego
* Rozkaz procesu testera – sprawdza czy jest adresatem rozkazu ( rozkazy create, kill i status ) i jeżeli tak wykonuje rozkaz i zapisuje jego wynik do potoku wyników. W przeciwnym wypadku przesyła rozkaz do procesu roboczego.

## Proces roboczy

Oczekuje w pętli na rozkazy od procesu testera. Po nadejściu rozkazu wysyła żądanie do procesu magazynującego i oczekuje na jego odpowiedź. Po odebraniu odpowiedzi na żądanie wysyła wynik i oczekuje na kolejny rozkaz. Kończy się gdy potok rozkazów zwróci wartość końca pliku.

# Komunikaty testujące

Komunikaty te są rozszerzeniem zbioru komunikatów podstawowych.

## Komunikaty rozkazów

Komunikat posiada typ określający wydawany rozkaz, numer rozkazu w celu późniejszej identyfikacji wyniku, pid procesu roboczego do którego rozkaz jest kierowany i dodatkowe parametry zależne od typu rozkazu. Rozkazy dla procesu magazynującego nie muszą zawierać pidu tego procesu.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rozmiar|int | Typ|int | NrRozkazu|int | Pid|int | Parametry |
| Rozmiar|int | **Typ|int** | **NrRozkazu|int** | **Parametry** | |

## Komunikaty wyników

Komunikat posiada numer rozkazu którego dotyczy, kod błędu oraz dodatkowe informacje mogące być przydatne w celu wykrycia źródła błędu. W przypadku gdy kod błędu posiada wartość NoError, rozkaz zakończył się sukcesem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rozmiar|int | Typ|int | NrRozkazu|int | KodBłędu|int | Dodatkowe informacje |

# Zarys implementacji

|  |  |
| --- | --- |
| **Język** | C++ |
| **Biblioteki** | Standardowa i STL |
| **Narzędzia** | IDE Netbeans z kompilatorem gcc |