Opis rozwiązania
3. zadania zaliczeniowego
z Programowania Współbieżnego

Piotr Nosek, ***pn371273***

1. Poczynione założenia

W trakcie projektowania i implementacji rozwiązania, poczynione zostały następujące założenia:

- procesów testera działających naraz nie może być więcej, niż możliwych do przydzielenia im numerów PID, ale także nie więcej niż liczba możliwych do utworzenia kolejek `mq` dzielona na 2
- serwer validator nie będzie przerywany sygnałami SIGKILL itp. Jedyna możliwość zatrzymania serwera to wysłanie znaku **!** przez jakiegoś testera
- weryfikacja słowa (czyli proces run) odbywa się współbieżnie - każde rozgałęzienie automatu powoduje odpalenie nowych procesów, które niezależnie liczą osobne gałęzie
- serwer validator działa współbieżnie - tworzy nowy podproces na każde otrzymane zapytanie od jakiegoś testera
- procesy tester działają sekwencyjnie - nie mają potrzeby działać współbieżnie, gdyż nie daje to istotnej korzyści. Wczytują kolejne słowa z wejścia, wysyłają do weryfikacji i odpytują kolejkę, na której pisze do nich serwer, czy nie przyszła jakaś odpowiedź - działa ona nieblokująco
- jeśli proces tester napotka błąd, to kończy działanie, ale nie powoduje to końca pracy serwera - może on działać nadal, niezależnie od błędów testerów

2. Opis komunikacji

Komunikacja odbywa się poprzez łącza nienazwane (pipe) oraz kolejki wiadomości mq. Wykorzystywane są również sygnały - sygnał `SIGRTMIN`.

- serwer validator komunikuje się z procesami tester przy pomocy kolejek mq. Jest jedna globalna kolejka **testers**, która złoży do rejestracji testerów na testerze - wysyłają tam one swoje numery PID. Po zarejestrowaniu, każdy tester otrzymuje dwie kolejki: r<PID> oraz w<PID>. Służą one odpowiednio do komunikacji serwer-tester oraz tester-serwer. Na kolejkę w<PID> tester wkłada wiadomości o słowach do weryfikacji. Serwer odczytuje je z tej kolejki,

a odpowiedź odsyła na kolejkę w<PID>, z której czyta tester o ID procesu równym PID.

- na tę samą kolejkę **testers** proces tester może wysłać również informację, że zakończył już działanie (wysyła wiadomość !<PID>) - jest uznawany wtedy za martwego testera i nie jest uwzględniany przy różnych działaniach przez serwer
- serwer validator uruchamia procesy run i komunikuje się z nimi za pomocą łącz nienazwanych pipe. Każdy główny proces run ma jednego pipe'a. Validator przekierowuje stdout i pisze na niego opis automatu oraz słowo do weryfikacji. Po zakończonej weryfikacji, proces run odsyła poprzez to łącze wynik weryfikacji - string "1" lub "0", reprezentujący poprawne bądź niepoprawne słowo.
- proces run uruchamia podprocesy run, z którymi komunikuje się za pomocą pipe'ów. Podproces run wysyła na łącze rodzica wiadomość "1", jeśli weryfikacja jego gałęzi zakończyła się sukcesem, bądź "0" w.p.p. Rodzic składa odpowiedzi od dzieci przy pomocy logicznej alternatywy bądź koniunkcji (w zależności od tego jakiego typu stanem jest) i przesyła odpowiedź wyżej, do swojego rodzica. Główny proces run przesyła gotową odpowiedź do serwera validator
- podprocesy validatora, które wczytują słowa od testerów, odpalają procesy run i czekają na zakończenie weryfikacji, komunikują się z głównym procesem validatora za pomocą łącz nienazwanych - wysyłają na nie wyniki zapytania, które potem validator zbiera, żeby wypisać podsumowanie swojej pracy. Zadanie tych podprocesów jest następujące:
    - odczytać odpowiedź od testera
    - odpalić odpowiedni proces run i wysłać mu słowo do weryfikacji
    - poczekać na odpowiedź
    - przekazać odpowiedź testerowi
- podczas czytania wiadomości od testerów, jedna z nich może być wiadomością kończącą pracę, tj. !. Gdy validator napotka taką wiadomość, to czeka aż wszystkie jego podprocesy zakończą pracę (wszystkie weryfikacje zakończą się, a odpowiedzi zostaną odesłane do pytających), a następnie wysyła do wszystkich aktywnych procesów tester sygnał `SIGRTMIN`. Po wysłaniu sygnałów proces zwalnia zasoby i kończy pracę.
- gdy procesy tester otrzymają sygnał `SIGRTMIN` czekają na otrzymanie wszystkich odpowiedzi (tyle, ile zapytań wysłali), zwalniają zasoby i kończą pracę

3. Wykorzystywane zasoby

Rozwiązanie wykorzystuje następujące zasoby:

- globalną kolejkę wiadomości (mq) testers
- po dwie kolejki mq dla każdego procesu tester
- łącze nienazwane dla każdego procesu tester (do komunikacji podprocesy validator - główny proces validator, każdy tester zbiera informacje ile zapytań do niego było i ile zostało zaakceptowanych)
- jedno łącze nienazwane, na którym podprocesy validator odkładają informację, że wysłały odpowiedź do testera (pole Snt w podsumowaniu validatora)
- łącze nienazwane dla każdego zapytania przesłanego przez testera (do komunikacji validator-run)
- łącze nienazwane dla każdego wierzchołka automatu, który ma więcej niż jedną krawędź wychodzącą dla pewnej etykiety - do komunikacji z dziećmi

Po zakończeniu działania, wszystkie zasoby są zwalniane - deskryptory zamykane, kolejki usuwane z sytemu plików. Zasoby są też zwalniane przy napotkaniu błędu podczas korzystania z funkcji systemowej, który uniemożliwiałby dalsze działanie rozwiązania - każdy program zawiera funcję `terminate()`.