[PW] Zadanie zaliczeniowe 3: Automaty

Aleksander Wiącek

1 Opis komunikacji międzyprocesowej i wykorzystanych mechanizmów

1. Proces `validator` rozpoczyna swoją pracę od otworzenia dwóch kolejek: `REQUESTS` (do odbierania zleceń od `tester`ów) oraz `RESULTS` (do odbierania wyników od `run`ów). Następnie wczytuje opis automatu i zapisuje go w innej formie:
   - Pierwsza linia zawiera informację o tym, ile stanów jest uniwersalnych.
   - Druga linia zawiera stany akceptujące lub znak `-`, jeśli takich nie ma.
   - Następne linie zawierają opis funkcji przejścia. Po wczytaniu są one sortowane.
2. Proces `tester` rozpoczyna swoją pracę od otworzenia kolejki `REQUESTS` i utworzenia kolejki `ANSWERS_PID`, w której będzie otrzymywał odpowiedzi od `validator`a. Gdy otrzyma ze standardowego wejścia słowo `word` do przetworzenia, wysyła `validator`owi komunikat postaci `pid word`, gdzie `pid` jest identyfikatorem testera. Jeśli wczyta znak `!`, przesyła znak `!` do `validator`a. Tuż przed zakończeniem swojego działania proces `tester` wysyła `validator`owi komunikat `. pid`, który oznajmia mu, że `tester` o podanym `pid` zakończył swoje działanie. Ostatni komunikat ma priorytet 1, pierwsze dwa 0.
3. Proces `validator` czeka na komunikaty z dwóch kolejek. Jeżeli odbierze komunikat z kolejki `REQUESTS`, to wykonuje jedną z poniższych czynności (w zależności od komunikatu i tego, czy pracuje):
   - Jeśli odbierze komunikat `. pid`, to zapamiętuje, że proces `tester` o numerze `pid` już nie pracuje.
   - Jeśli pracuje i odbierze komunikat `!`, to od tej pory nie pracuje i wysyła wszystkim pracującym[1] `tester`om komunikat `!` z priorytetem 1 (kolejki `ANSWERS_PID` są otwierane i zamykane na bieżąco).
   - Jeśli nie pracuje, ale jeszcze czeka na procesy `run` i odbierze zlecenie od `tester`a, to odsyła mu komunikat `?` z priorytetem 0.
   - Jeżeli pracuje i otrzymuje standardowe zlecenie, to tworzy łącze nienazwane i wykonuje `fork`a. Dziecko przygotowuje parametry dla procesu `run` (o nich niżej) i wywołuje go (`execl`), a rodzic aktualizuje zmienne lokalne i podaje dane procesowi `run` przez łącze, po czym wraca do głównej pętli.

   Jeżeli odbierze komunikat z kolejki `RESULTS`, to aktualizuje lokalne zmienne i przekazuje wynik odpowiedniemu `tester`owi.
4. Proces `run` jest wywoływany z następującymi parametrami:
   - `pid` procesu `tester`, który zlecił przetworzenie tego słowa,
   - miejsce, w które należy przesłać rozwiązanie (-1 oznacza kolejkę `RESULTS`, inna liczba deskryptor),
   - stan, w którym znajduje się automat,
   - rozmiar opisu automatu (w liniach),
   - deskryptor, z którego proces otrzyma opis automatu,
   - słowo do rozpatrzenia (uwaga: słowo zawsze jest zakończone znakiem nowej linii; w szczególności puste słowo zawiera tylko znak nowej linii).
5. Proces `run` rozpoczyna swoją pracę od wczytania opisu automatu. Po tym przetwarza słowo na jeden z dwóch sposobów:
   - Jeśli słowo jest puste, to patrzy na drugą linię wczytanego automatu i sprawdza, czy jest tam stan, w którym obecnie znajduje się automat. W zależności od tego wysyła wynik `A` lub `N`.
   - Jeśli słowo jest niepuste, to szuka w opisie funkcji przejścia linii odpowiadającej obecnemu stanowi i pierwszej literze słowa.[2] Jeśli takiego nie ma, to `run` zwraca wartość `W`.[3] W przeciwnym razie dla każdego stanu należącego do `T(q,a)` (`q` jest obecnym stanem, zaś `a` pierwszą literą rozpatrywanego słowa) wykonuje następujące czynności:
     - Tworzy dwa łącza nienazwane - jedno będzie służyć do odebrania wyników, a drugie do przekazania automatu.
     - Wykonuje `fork`a. Dziecko przygotowuje parametry dla kolejnego procesu `run` i wywołuje go, a rodzic podaje przez utworzone łącze opis automatu.

[1] Proces `tester` jest oznaczany jako pracujący, gdy `validator` odbierze jego pierwsze zgłoszenie.
[2] Uwaga dotycząca złożoności: ponieważ `validator` posortował opis funkcji przejścia, to możemy tu skorzystać z wyszukiwania binarnego.
[3] Patrz ostatni punkt Założeń i wyjaśnień.

Następnie czeka na wyniki wszystkich wywołanych `run`ów i liczy alternatywę otrzymanych wyników (jeśli automat jest w stanie egzystencjalnym) lub ich koniunkcję (jeśli jest w stanie uniwersalnym).

6. Komunikaty wysyłane przez procesy `run` do kolejki `RESULTS` są postaci `pid word R`, gdzie `pid` jest identyfikatorem procesu `tester`, `word` jest zleconym słowem, a `R` oznacza wynik i jest literą `A` lub `N`.

2 Założenia i wyjaśnienia

1. Zakładam, że działa co najwyżej jeden proces `validator`.

2. Proces `validator` zakłada, że `pid` testera jest mniejszy niż `PID_MAX = 49152`. Wartość ta została wzięta z `/proc/sys/kernel/pid_max`. Ponadto `validator` nie rozróżnia dwóch różnych testerów o tym samym `pid` (gdyby np. jeden z nich zakończył pracę, a potem powstał drugi o takim samym identyfikatorze, to `validator` zsumuje ich zlecenia w raporcie końcowym).

3. W przypadku, gdy `tester` wyśle zlecenie `validator`owi, który otrzymał już komunikat `!` (może się tak zdarzyć, gdy `tester` rozpocznie swoją pracę po tym, jak `validator` otrzymał komunikat `!` lub gdy `validator` wyśle komunikat o zakończeniu pracy *mniej więcej* w tym samym momencie, co `tester` swoje zapytanie) nastąpi jedna z trzech poniższych sytuacji:

   - Proces `validator` jeszcze działa, bo oczekuje na wyniki od procesów `run`. W tym wypadku odeśle on `tester`owi komunikat `?`, który należy traktować jako odrzucenie zlecenia - `tester` notuje, że wysłał to zlecenie, ale nie dostał odpowiedzi (zmienna `fake_received`).
   - Proces `validator` już zakończył pracę (nie licząc sprzątania i wypisania raportu) - `tester` najprawdopodobniej się zawiesi, bo oczekuje na wynik zlecenia, a nie dostał komunikatu `?` odrzucenia zlecenia.
   - Nigdy nie uruchomiliśmy `validatora`. Wtedy `tester` zwróci błąd przy próbie otwarcia nieistniejącej kolejki `REQUESTS`.

4. Procesy korzystają z sygnałów do przerywania pracy: proces, w którym wystąpił błąd, wysyła sygnał `SIGKILL` `validator`owi (oczywiście, `validator` sam sobie sygnału nie wysyła). Ten natomiast rozsyła sygnał `SIGKILL` do wszystkich działających `tester`ów i `run`ów.

5. W przypadku, gdy procesy kończą pracę z powodu sygnału, nie zamykam żadnych łącz nienazwanych. Robi to za mnie system, gdy proces kończy swoją pracę z powodu sygnału.

6. Gdy procesowi `tester` lub `validator` pozostanie tylko posprzątanie po sobie oraz wypisanie wyjścia, to w przypadku, gdyby któraś z operacji (związanych ze sprzątaniem: `mq_close`, `mq_unlink` oraz `mq_send(REQUESTS, ". pid", ...)` w `tester`ze) się nie udała, ignorujemy błąd.

7. Trójwartościowa logika procesów `run`:

   - Z praktycznego punktu widzenia wynik `W` uzyskany przez proces `run` oznacza "nie istnieje ścieżka, za pomocą której automat przetworzy całe słowo".
   - Wartość `W` w operacjach logicznych jest traktowana jako `NULL` (ale nie jako nie-wiadomo-co, tylko faktycznie pusta wartość). Innymi słowy, `W AND x = W OR x = x` dla dowolnego `x`.
   - Gdyby `run` miał zwrócić wartość `W` `validator`owi, to zamiast tego zwróci `N`. Skoro bowiem nie istnieje żaden bieg automatu na słowie o oczekiwanej długości, to tym bardziej nie istnieje bieg akceptujący.