



Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Implementacja maszyny wirtualnej dla funkcyjnych języków
programowania wspierających przetwarzanie współbieżne.*

*Implementation of a virtual machine for functional programming
languages with support for concurrent computing.*

Autor:	<i>Kajetan Rzepecki</i>
Kierunek studiów:	<i>Informatyka</i>
Opiekun pracy:	<i>dr inż. Piotr Matyasik</i>

Kraków, 2014

*Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nie-
prawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie
i nie korzystałem ze źródeł innych niż wymienione w pracy.*

Serdecznie dziękuję Lucynie za cierpliwość i wsparcie podczas tworzenia pracy.

Spis treści

1. Model przetwarzania współbieżnego	7
1.1. Implementacja symetrycznego multiprocessingu	7
1.2. Harmonogramowanie procesów	10
1.3. Implementacja Modelu Aktorowego	11
1.4. Implementacja przesyłania wiadomości	12

1. Model przetwarzania współbieżnego

Niniejsza sekcja opisuje implementację modelu przetwarzania współbieżnego zastosowanego w maszynie wirtualnej ThesisVM. Model ten przewiduje wykorzystanie symetrycznego multiprocessingu oraz implementację Modelu Aktorowego interakcji mikroprocesów. Wstępny opis wybranego modelu zawarto w sekcji ??.

Model Aktorowy [?] został wybrany przez wzgląd na jego relatywne nieskomplikowanie i wielką ekspresywność, zwłaszcza w kontekście funkcyjnych języków programowania. Model ten zakłada istnienie autonomicznych *aktorów*, którzy porozumiewają się za pomocą przekazywanych asynchronicznie *wiadomości*, co bardzo łatwo można przetłumaczyć na istnienie wielu działających konkurencyjnie mikroprocesów.

Każdy mikroproces po otrzymaniu wiadomości może na nie reagować poprzez zmianę swojego wewnętrznego *zachowania*, wygenerowanie skończonej liczby *nowych wiadomości*, wysyłanych konkurencyjnie do innych mikroprocesów, lub poprzez stworzenie skończonej liczby *nowych aktorów* - uruchomienie dodatkowych mikroprocesów ThesisVM. Dokładny opis założeń Modelu Aktorowego został zawarty w [?].

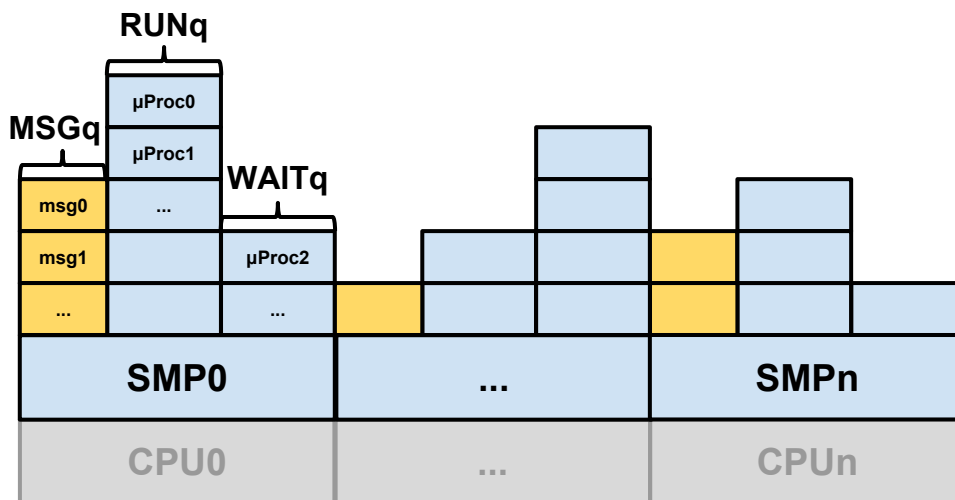
1.1. Implementacja symetrycznego multiprocessingu

Implementacja wykorzystuje wątki systemu operacyjnego, na którym uruchomiona jest maszyna wirtualna ThesisVM w celu zrównoleglenia działania wielu interpreterów kodu bajtowego. Rysunek 1.1 zawiera schematyczną reprezentację struktury symetrycznych multiprocesorów (SMP).

SMP komunikują się ze sobą poprzez wiadomości kontrolne przekazywane za pośrednictwem kolejki wiadomości `MSGq`, efektywnie wykorzystując Model Aktorowy. Implementacja taka jest więc bardzo skalowalna i umożliwia dowolną zmianę ilość uruchomionych jednostek także podczas działania maszyny wirtualnej.

Obecnie wiadomości kontrolne wykorzystywane są przy tworzeniu nowych mikroprocesów, ale implementacja może zostać w przyszłości rozszerzona w celu umożliwienia stosowania zaawansowanych algorytmów równoważenia obciążenia, strategii uruchomie-

niowych mikroprocesów a także propagacji i kolekcji danych diagnostycznych działania symetrycznych multiprocesorów. Więcej informacji o przyszłych kierunkach rozwoju projektu zostało zawarte w sekcji ??.



Rysunek 1.1: Schemat symetrycznego multiprocessingu ThesisVM.

Każdy z symetrycznych multiprocesorów (SMP) zarządza szeregiem struktur danych wykorzystywanych do przechowywania kontekstów mikroprocesów oraz harmonogramowania (ang. *scheduling*) ich interpretacji (rysunek ref:fig:tvm-smp).

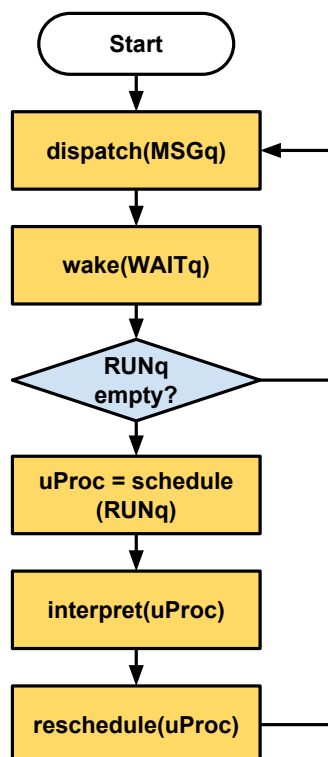
Struktury te to kolejki priorytetowe zaimplementowane w oparciu o, gwarantujące logarytmiczną złożoność wszystkich operacji, **drzewa czerwono-czarne**:

- RUNq - kolejka ustalającą kolejność uruchamiania aktywnych mikroprocesów,
- WAITq - kolejka przechowująca mikroprocesy będące w uśpieniu, ułatwiająca ustalenie kolejności ich powrotu do działania.

SMP działają według uproszczonego algorytmu zaprezentowanego na diagramie 1.2. Po starcie maszyny wirtualnej wszystkie SMP przechodzą do fazy obsługi wiadomości kontrolnych, **dispatch**. Podczas tej fazy wiadomości otrzymane asynchronicznie przez SMP są analizowane i obsługiwane tak szybko, jak to tylko możliwe.

Następną fazą jest faza **wake**, której zadaniem jest przywrócenie uśpionych mikroprocesów do ponownego działania. Wykorzystuje ona kolejkę WAITq, dzięki czemu ustalenie, czy istnieją mikroprocesy gotowe do przebudzenia mogło zostać zrealizowane w czasie logarytmicznym poprzez analizę wartości skrajnie lewego poddrzewa reprezentacji kolejki, gdzie znajduje się element najmniejszy.

Jeśli jakiegokolwiek mikroprocesy zostały przebudzone i tym samym przeniesione do kolejki RUNq, następuje faza ich harmonogramowania - **schedule**. Faza ta ustala kolejność uruchamiania i interpretacji kodu poszczególnych mikroprocesów i została szczegółowo opisana w następnej sekcji. Jeśli żaden mikroproces nie oczekuje na uruchomienie SMP wraca do pierwszej fazy oczekując na nowe wiadomości kontrolne.



Rysunek 1.2: Algorytm postępowania symetrycznych multiprocessorów ThesisVM.

Po ustaleniu mikroprocesu gotowego do uruchomienia następuje faza interpretacji jego kodu - **interpret**. Faza ta jest ograniczona czasowo (ang. *time-based scheduling*) a czas jej trwania zależy od obecnego obciążenia SMP. Alternatywnym rozwiązaniem jest ograniczenie maksymalnej ilości kroków interpretera kodu bajtowego (ang. *work-based scheduling*).

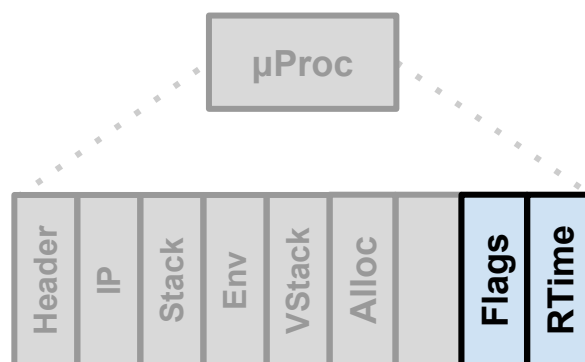
Ostatnia faza, **reschedule** polega na wywłaszczeniu mikroprocesu i przeniesieniu go do jednej z kolejek RUNq lub WAITq w zależności od efektów jego działania. Po tej fazie następuje koniec cyklu i symetryczny multiprocessor ponownie przechodzi do fazy obsługi wiadomości kontrolnych.

1.2. Harmonogramowanie procesów

Najważniejszą fazą opisanego w poprzedniej sekcji algorytmu jest faza harmonogramowania procesów. W celu ustalenia kolejności mikroprocesów wykorzystanie został algorytm **Completely Fair Scheduling** (CFS), który jest stosowany między innymi w jądrze systemu Linux od wersji 2.6.23.

Kluczową cechą algorytmu CFS jest wykorzystanie **wirtualnych czasów** działania zadań, które obliczane są w różny sposób w zależności od priorytetu zadania, co pozwala na wykorzystanie jednej kolejki do harmonogramowania procesów o różnych priorytetach, zamiast wielu osobnych kolejek, dla różnych priorytetów.

Wirtualne czasy działania zadań przechowywane są osobno dla każdego zadania i są modyfikowane po każdym cyklu ich uruchomienia. Rysunek 1.3 zawiera schemat rozmieszczenia dodatkowych rejestrów mikroprocesów ThesisVM koniecznych do zaimplementowania algorytmu CFS.



Rysunek 1.3: Schemat rejestrów wymaganych przez usprawnienia harmonogramowania SMP.

Rejestr **Flags** przechowuje informację o priorytecie mikroprocesu, a rejestr **RTime** o dotychczasowym, *rzeczywistym* czasie jego działania. Wartość czasu *wirtualnego* wyznaczana jest jako iloczyn priorytetu i rzeczywistego czasu działania.

W każdym cyklu działania SMP mikroproces o najniższej wartości wirtualnego czasu działania pobierany jest z kolejki **RUNq**. Operacja ta, podobnie jak analogiczna operacja dotycząca uśpionych mikroprocesów wykonywana jest w czasie logarytmicznym dzięki wykorzystaniu drzew czerwono czarnych w implementacji kolejki **RUNq**.

Dla tak desygnowanego procesu obliczany jest czas dostępu do procesora (ang. *fair share*), który zależy od konfigurowalnej wartości maksymalnej oraz ilości mikroprocesów aktualnie oczekujących na uruchomienie. Zapewnia to zwiększenie *interaktywności* mikroprocesów kosztem zwiększenia liczby zmian ich kontekstów.

Nowo utworzone mikroprocesy, a także te reaktywowane po czasie uśpienia dodawane są do kolejki RUNq z aktualnie minimalną wartością wirtualnego czasu działania. Technika ta nosi miano **sleeper fairness** i gwarantuje, że mikroprocesy, które przez dłuższy czas były w stanie uśpienia otrzymają porównywalny udział czasu procesora sprzętowego.

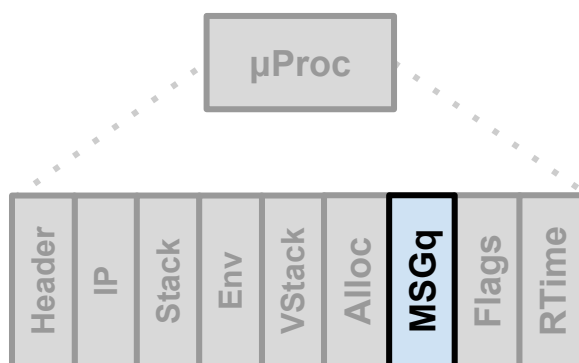
Obecna implementacja harmonogramowania nie wykorzystuje niestety algorytmów równoważenia obciążenia. W przyszłości może zostać jednak rozwinięta umożliwiając podział kolejki RUNq i przekazanie części przynależących do niej mikroprocesów do pozostałych symetrycznych multiprocessorów.

1.3. Implementacja Modelu Aktorowego

Implementacja Modelu Aktorowego w maszynie wirtualnej ThesisVM objawia się wykorzystaniem autonomicznych mikroprocesów, które porozumiewają się za pomocą asynchronicznego przekazywania wiadomości.

Mikroprocesy tworzone są za pomocą prymitywnej operacji **spawn**, która jako argumenty przyjmuje nazwę funkcji startowej nowego mikroprocesu oraz argument, który zostanie jej przekazany. Tworzony jest wtedy nowy kontekst mikroprocesu, którego rejestry zostają odpowiednio spreparowane by umożliwić natychmiastowe jego wykonanie. Kontekst jest następnie przekazywany do jednego z symetrycznych multiprocessorów, gdzie rozpoczyna swoje działanie.

Działające współbieżnie mikroprocesy mogą komunikować się za pośrednictwem prymitywnych operacji **send** oraz **recv**, które odpowiednio inicjalizują wysyłanie wiadomości i przechwytują następną wiadomość z kolejki wiadomości MSGq mikroprocesu. Kolejka ta przechowywana jest w osobnym rejestrze kontekstu mikroprocesu; rysunek 1.4 zawiera schemat położenia rejestru MSGq w obiekcie reprezentującym mikroproces.

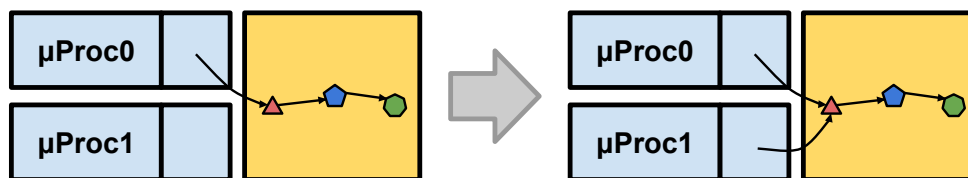


Rysunek 1.4: Schemat rejestrów wymaganych przez implementację Modelu Aktorowego.

Implementacja zapewnia *logiczną separację* pamięci procesów, ponieważ *nie wspiera mutacji* danych, co gwarantuje, iż jedynym sposobem ich współdzielenia między aktorami jest przekazywanie wiadomości. Należy zwrócić uwagę na fakt, że nie jest to *separacja fizyczna*. Dane przesyłane pomiędzy mikroprocesami egzystują w jednej puli pamięci i nie są w żaden sposób kopiowane.

1.4. Implementacja przesyłania wiadomości

Asynchroniczne przesyłanie wiadomości zostało zrealizowane z wykorzystaniem kolejek nieblokujących (ang. *non-blocking queue*) zaimplementowanych zgodnie z algorytmem Michael'a i Scott'a [?].



Rysunek 1.5: Schemat działania przesyłania wiadomości.

- Opisać sposób porozumiewania się procesów (kolejki nieblokujące). [?, ?]
- Wspomnieć o gwarancjach jakie daje algorytm. [?, ?]
- Opisać wykorzystanie CAS i problem ABA.
- Opisać krótko wady i możliwe usprawnienia zastosowanego rozwiązania (dynamic size, wait-free, optimistic FIFO). [?, ?, ?]
- Opisać krótko alternatywne podejścia (synchroniczne przekazywanie wiadomości - kanały, locki/mutexy/semafony).
- Opisać sposób pobierania wiadomości z kolejki i jego możliwe usprawnienia (pattern-matching).