Multiprocessor Shared-Memory Information Exchange

Damian Klata, Adam Bułak

Wstęp

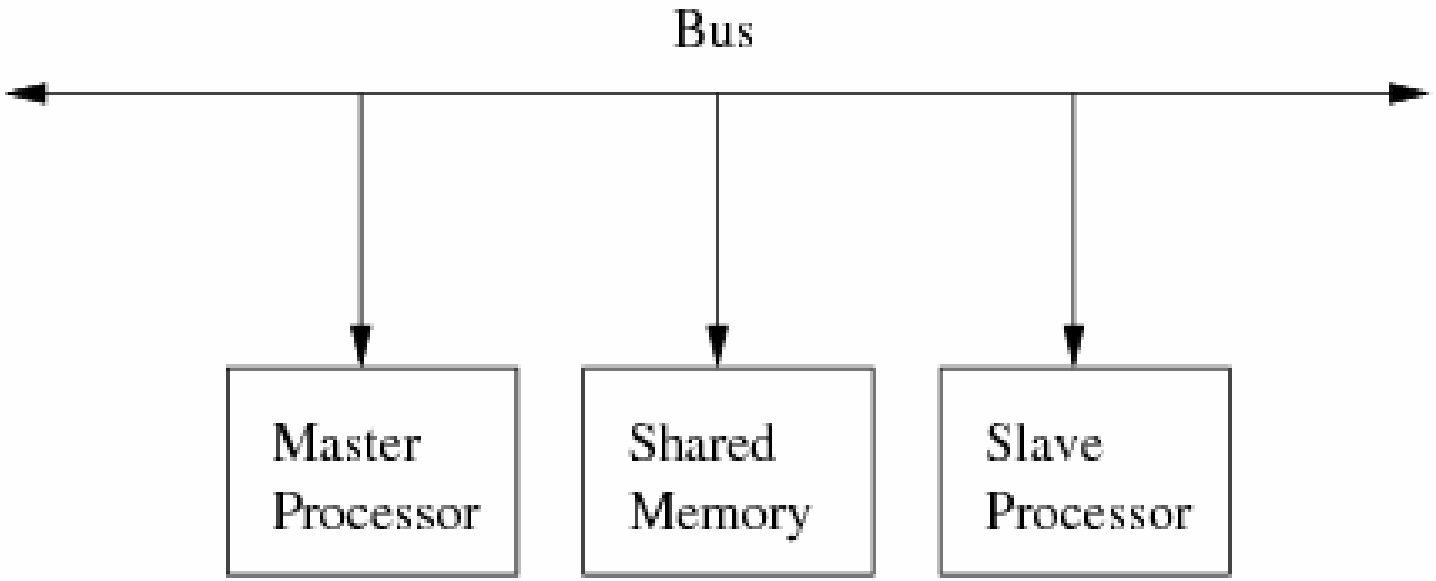
Zajmiemy się analizą protokołu opartego na komunikacji przez pamięć dzieloną opracowany przez firmę Westinghouse.

Protokół jest wykorzystywany w systemie komputerowym nadzorującym pracę reaktora atomowego Sizewell B w Wielkiej Brytanii.

Opis protokołu

- Dwa rodzaje procesorów: slave i master
- Każda komórka pamięci wykorzystywana do komunikacji tylko w jednym kierunku
- Komunikacja przez pamięć dzieloną udostępnianą przez wspólną szynę
- Czytanie i pisanie odbywa się asynchronicznie

A teraz obrazek...

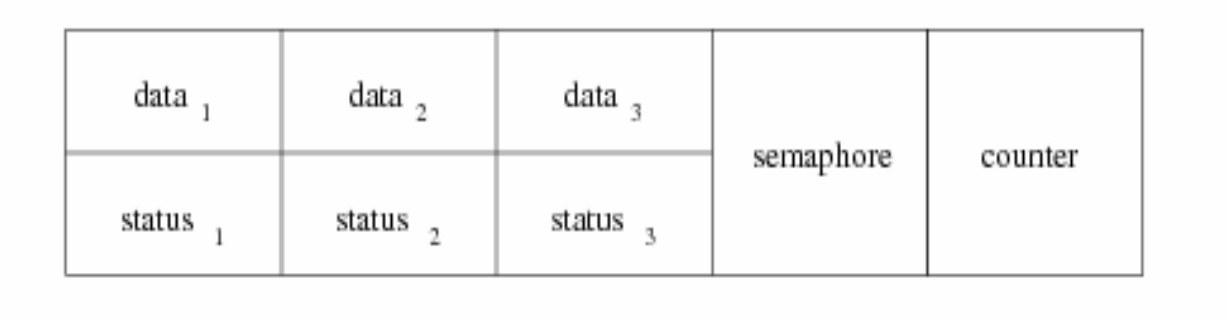


Wymagania

- Uniknięcie odczytywania danych, które nie zostały do końca uaktualnione (ang. data tearing).
- Przerwy w komunikacji muszą być możliwie jak najmniejsze i o ustalonej długości (właśnie dlatego komunikacja oparta na semaforze nie wchodzi w grę).

Jak to działa?

- Pamięć podzielona jest na jednostki składające się z:
  - trzech buforów – każdy składa się z pola opisującego jego stan i z pola danych
  - semafora
  - licznika czytelników



Możliwe stany buforów

- Każdy bufor przyjmuje jeden z czterech stanów:
  - **newest** – bufor zawiera najnowsze dane
  - **master** – bufor przeznaczony do czytania
  - **slave** – bufor przeznaczony do pisania
  - **idle** – bufor bezczynny

- Początkowo jeden bufor jest w stanie **slave**, a dwa pozostałe w stanie **idle**.

Operacja slave

1. Weź semafor.
2. Jeżeli jest bufor w stanie **newest**, zmień jego stan na **idle**.
3. Jeżeli jest bufor w stanie **slave**, zmień na **newest**, wpp błąd.
4. Jeżeli jest bufor w stanie **idle**, zmień jego stan na **slave**, wpp błąd.
5. Oddaj semafor.

Operacja master acquire

1. Weź semafor.
2. Jeżeli jest bufor w stanie **newest** i nie ma bufora w stanie **master**, zmień stan bufora **newest** na **master**.
3. Jeżeli jest bufor w stanie **master**, zwiększ liczbę czytelników.
4. Oddaj semafor.

Operacja master release

1. Weź semafor.
2. Jeżeli nie ma czytelników, to jeśli jest bufor w stanie **newest**, zmień stan bufora master na **idle**, wpp zmień stan bufora master na **newest**.
3. Oddaj semafor.

Model naturalny

- Model umożliwia sprawdzenie poprawnej koordynacji buforów. Nie pozwala sprawdzić stanu danych przechowywanych w pamięci ani operacji czytania/pisania pomiędzy akcjami protokołu.
- Model dostarcza przez agentów usługi procesorom. Jeśli procesor slave chce wykonać pewną akcję synchronizuje się z agentem na odpowiednim porcie.
- Interfejs agenta składa się z akcji:
  - **write** (port na którym komunikuje się procesor slave, gdy chce pisać)
  - $\mathbf{ma}_k$ (master aquire)
  - $\mathbf{mr}_k$ (master release)

Model obrazu pamięci

- Bufor przechowuje stan pamięci i pozwala na jego zapis i odczyt:

$$Buf(v) \stackrel{def}{=} \overline{\mathbf{read}}(v).Buf(v) + \mathbf{write}(v').Buf(v')$$

- Buforom nadajemy adresy przez przemianowanie portów związanych z pisaniem i czytaniem:

$$Buf_i(v) \stackrel{def}{=} Buf[\mathbf{read}_i/\mathbf{read}, \mathbf{write}_i/\mathbf{write}]$$

- Łączymy trzy bufory równolegle. Stany **idle**, **master**, **slave** i **newest** oznaczamy odpowiednio **i**, **m**, **s**, **n**:

$$Buffers \stackrel{def}{=} Buf_1(\mathbf{s}) \mid Buf_2(\mathbf{i}) \mid Buf_3(\mathbf{i})$$

Licznik i semafor

- Licznik definiujemy następująco:

$$\begin{aligned} Counter(n) \stackrel{def}{=} \quad & (\text{if } n = 0 \text{ then } \mathbf{zero}.Counter(n)) + \\ & + (\text{if } n < 2 \text{ then } \mathbf{inc}.Counter(n+1)) + \\ & + (\text{if } n > 0 \text{ then } \mathbf{dec}.Counter(n-1)) \end{aligned}$$

- Semafor zaś tak:

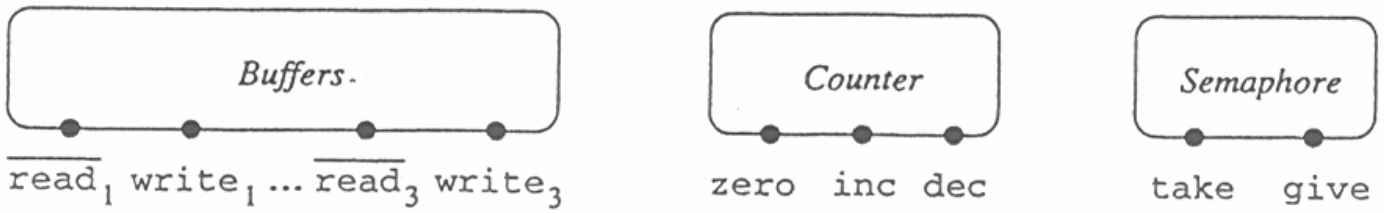
$$Semaphore \stackrel{def}{=} \mathbf{take}.\mathbf{give}.Semaphore$$

Obraz pamięci

- Obraz pamięci otrzymujemy w wyniku równoległego złożenia buforów, licznika i semafora:

$$MemImage \stackrel{def}{=} Buffers \mid Counter(0) \mid Semaphore$$

- A tak to wygląda agent *MemImage* z zewnątrz:



Agent Slave

$$
\begin{aligned}
Slave &\stackrel{def}{=} \mathbf{slave}.Sl_1 \\
Sl_1 &\stackrel{def}{=} \overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).Sl_2((v_1,v_2,v_3)) \\
Sl_2(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{n} \text{ then } Sl_3(B[i \to \mathbf{i}]) + \\
&\quad + \text{if } \mathbf{n} \notin \{B\#1, B\#2, B\#3\} \text{ then } Sl_3(B) \\
Sl_3(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{s} \text{ then } Sl_4(B[i \to \mathbf{n}]) \\
Sl_4(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{i} \text{ then } Sl_5(B[i \to \mathbf{s}]) \\
Sl_5(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.Slave
\end{aligned}
$$

Agent Master

$$
\begin{aligned}
Master &\stackrel{def}{=} MA_1 \\
MA_1 &\stackrel{def}{=} \overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).MA_2((v_1,v_2,v_3)) \\
MA_2 &\stackrel{def}{=} \text{if } \mathbf{n} \in \{B\#1, B\#2, B\#3\} \wedge \mathbf{m} \notin \{B\#1, B\#2, B\#3\} \\
&\quad \text{then } \sum_{i=1}^{3} (\text{if } B\#i = \mathbf{n} \text{ then } MA_3(B[i \to \mathbf{m}])) \\
&\quad \text{else } MA_3(B) \\
MA_3(B) &\stackrel{def}{=} (\text{if } \mathbf{m} \in \{B\#1, B\#2, B\#3\} \text{ then } \mathbf{ma}.\overline{\mathbf{inc}}.MA_4(B)) + \\
&\quad + \overline{\mathbf{give}}.Master \\
MA_4(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.MR_1 \\
MR_1 &\stackrel{def}{=} \mathbf{mr}.\overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).MR_2((v_1,v_2,v_3)) \\
MR_2(B) &\stackrel{def}{=} \overline{\mathbf{dec}}.MR_3(B) \\
MR_3(B) &\stackrel{def}{=} \overline{\mathbf{zero}}.\sum_{i=1}^{3} \Big(\text{if } B\#i = \mathbf{m} \text{ then} \\
&\quad (\text{if } \mathbf{n} \in \{B\#1, B\#2, B\#3\} \text{ then } MR_4(B[i \to \mathbf{i}]) \text{ else } MR_4(B[i \to \mathbf{n}])\Big) + \\
&\quad + \overline{\mathbf{dec}}.\overline{\mathbf{inc}}.MR_4(B) \\
MR_4(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.Master
\end{aligned}
$$

Model abstrakcyjny

- Badamy tylko obserwowalne zachowania systemu
- Model prostszy
- Sekwencyjny
- Sparametryzowany obrazem pamięci
- Definiujemy jako agenta $M(B, r)$. Parametr B jest zbiorem trzech par (a, l), gdzie:
  - a to stan bufora (ze zbioru {**i**, **s**, **m**, **n**})
  - l to identyfikator bufora (ze zbioru {1, 2, 3})
  - R to zbiór czytelników (podzbiór zbioru {1, 2} identyfikatorów procesorów master
- Opis agenta M podajemy w formie dopuszczalnych tranzycji.

Akcje procesora slave

$$M(\{(\mathbf{s},l),(\mathbf{n},l'),(\mathbf{i},l'')\},r) \xrightarrow{\text{slave}} M(\{(\mathbf{n},l),(\mathbf{s},l'),(\mathbf{i},l'')\},r) \quad (1)$$

$$M(\{(\mathbf{s},l),(\mathbf{n},l'),(\mathbf{i},l'')\},r) \xrightarrow{\text{slave}} M(\{(\mathbf{n},l),(\mathbf{i},l'),(\mathbf{s},l'')\},r) \quad (2)$$

$$M(\{(\mathbf{s},l),(\mathbf{i},l')\} \cup B',r) \xrightarrow{\text{slave}} M(\{(\mathbf{n},l),(\mathbf{s},l')\} \cup B',r) \quad \forall l.(\mathbf{n},l) \notin B' \quad (3)$$

$$M(\{(\mathbf{s},l),(\mathbf{n},l')\} \cup B',r) \xrightarrow{\text{slave}} M(\{(\mathbf{n},l),(\mathbf{s},l')\} \cup B',r) \quad \forall l.(\mathbf{i},l) \notin B' \quad (4)$$

Akcje procesora master

- Master acquire:

For $k \in MI$:

$$M(\{(\mathrm{m},l)\} \cup B', r) \xrightarrow{\mathrm{ma}_k} M(\{(\mathrm{m},l)\} \cup B', r \cup \{k\}) \quad k \notin r \qquad (5)$$

$$M(\{(\mathrm{n},l)\} \cup B', \emptyset) \xrightarrow{\mathrm{ma}_k} M(\{(\mathrm{m},l)\} \cup B', \{k\}) \quad \forall l.(\mathrm{m},l) \notin B' \qquad (6)$$

- Master release:

For $k \in MI$:

$$M(\{(\mathrm{m},l)\} \cup B', r\{\cup k\}) \xrightarrow{\mathrm{mr}_k} M(\{(\mathrm{m},l)\} \cup B', r) \quad r \neq \emptyset, k \notin r \qquad (7)$$

$$M(\{(\mathrm{m},l),(\mathrm{n},l')\} \cup B', \{k\}) \xrightarrow{\mathrm{mr}_k} M(\{(\mathrm{i},l),(\mathrm{n},l')\} \cup B', \emptyset) \qquad (8)$$

$$M(\{(\mathrm{m},l)\} \cup B', \{k\}) \xrightarrow{\mathrm{mr}_k} M(\{(\mathrm{n},l)\} \cup B', \emptyset) \quad \forall l.(\mathrm{n},l) \notin B' \qquad (9)$$

Agent Msmie’

- Agent Msmie’ to agent z buforami zainicjowanymi na **s**, **i**, **i**:

$$Msmie' \stackrel{\text{def}}{=} M(\{(\mathtt{s},1),(\mathtt{i},2),(\mathtt{i},3)\},\emptyset)$$

- Abstrakcyjny model może teraz zostać opisany jako agent

$$M(B,r) \stackrel{\text{def}}{=} R_1 + \cdots + R_9$$

gdzie agent *Ri* realizuje *i-tą* regułę, np. agent *R3* byłby zdefiniowany tak:

$$\sum_{(l,l',B')\in S} \mathtt{slave}\,.M(\{(\mathtt{n},l),(\mathtt{s},l'),(\mathtt{i},l'')\},r)$$

gdzie zbiór *S* zawiera wszystkie krotki (*l, l’, B’*) taka że $B = \{(\mathrm{s}, l), (\mathrm{i}, l')\} \cup B'$ nie zawiera ani jednego bufora o statusie **n**.

Analiza poprawności protokołu

- Przedstawione modele są w słabej bisymulacji (CWB). Jest między nimi jednak istotna różnica. W modelu naturalnym może się zdarzyć nieskończony ciąg sprawdzeń obrazu pamięci.
- Do sprawdzania poprawności działania buforów użyjemy modelu *Msmie'*. Można bowiem w nim sprawdzić, czy np. model się blokuje lub czy istnieją jakieś ciągi akcji, które nigdy nie zostaną wykonane (a chcielibyśmy, żeby wykonane zostały).

Weryfikacja wymagań

- Pierwsze wymaganie o spójności danych nie może zostać zweryfikowane w naszym modelu. Z opisu nieformalnego widać jednak, że własność zachodzi: procesor master może czytać tylko z bufora **m**, do którego nie może pisać slave.
- Drugie wymaganie o krótkich i o ustalonej długości opóźnieniach można wyrazić następującą formułą:

$$Always(\langle\langle \mathbf{slave} \rangle\rangle \mathbf{tt}),$$

  która mówi, że akcja **slave** może zostać zawsze wykonana. *Msmie'* spełnia tę formułę.
- Bufor powinien też być zawsze dostępny do odczytu dla procesora master (przynajmniej od chwili przejścia pod semaforem):

$$[[\mathbf{slave}]] Always(\langle\langle \mathbf{ma_1}, \mathbf{mr_1} \rangle\rangle \mathbf{tt} \wedge \langle\langle \mathbf{ma_2}, \mathbf{mr_2} \rangle\rangle \mathbf{tt})$$

Weryfikacja wymagań, cd.

- Chcielibyśmy sprawdzić, czy wartość którą zapisał procesor slave zostanie w końcu odczytana przez jakiś procesor master.
- Musimy poprawić model.
- Zmieniamy regułę 6:
  - stara reguła

$$M(\{(\mathrm{n}, l)\} \cup B', \emptyset) \xrightarrow{\mathrm{ma}_k} M(\{(\mathrm{m}, l)\} \cup B', \{k\}) \qquad \forall l.(\mathrm{m}, l) \notin B'$$

  - nowa reguła:

$$M(\{(\mathrm{n}, l)\} \cup B', \emptyset) \xrightarrow{\mathrm{ma}'_k} M(\{(\mathrm{m}, l)\} \cup B', \{k\}) \quad \forall l.(\mathrm{m}, l) \notin B'$$

Wyprowadzenie formuły

- Rozważmy agenta o typie $\{a, b, c\}$
- a zawsze zajdzie:

$$\mu X.\langle - \rangle \, \mathtt{tt} \wedge [b, c]X$$

- Dla słabych operatorów:

$$\mu X.\langle\!\langle - \varepsilon \rangle\!\rangle \, \mathtt{tt} \wedge [\![b, c]\!]X$$

- Zabraniamy tylko nieskończonych ciągów b w których nie występuje a

$$\mu X.\nu Y.\langle\!\langle - \varepsilon \rangle\!\rangle \, \mathtt{tt} \wedge [\![c]\!]Y \wedge [\![b]\!]X$$

- Generalizujemy – a musi wystąpić jeśli b_1 i b_2 będą się ciągle pojawiać:

$$\mu X.\nu Y_1.\langle\!\langle - \varepsilon \rangle\!\rangle \, \mathtt{tt} \wedge [\![c, b_2]\!]Y_1 \wedge [\![b_1]\!]$$
$$\nu Y_2.\langle\!\langle - \varepsilon \rangle\!\rangle \, \mathtt{tt} \wedge [\![c, b_1]\!]Y_2 \wedge [\![b_2]\!]X$$

Wyprowadzenie formuły, cd.

- Teraz możemy sformułować regułę mówiącą, że *ma*$_k$*'* zajdzie jeśli obie akcje *ma*$_k$
- będą występować:

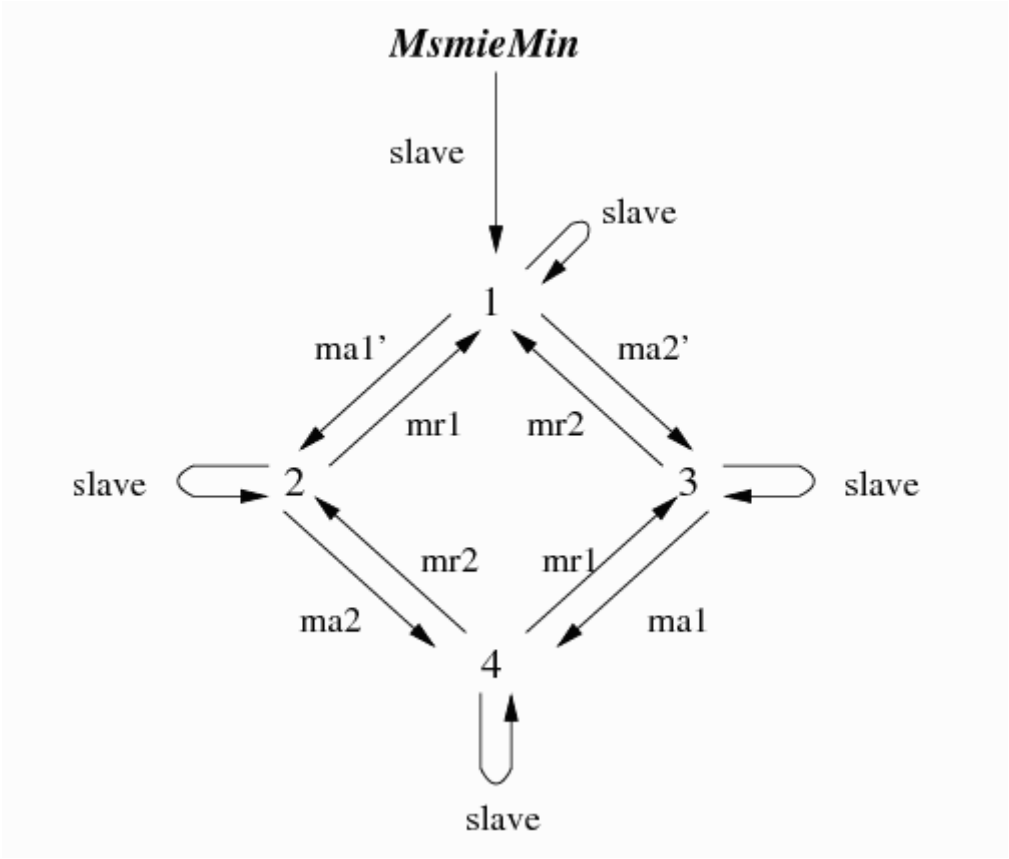
$$\mu X.\nu Y_1.\langle\langle -\varepsilon\rangle\rangle \mathtt{tt} \wedge [\![\mathtt{mr}_1, \mathtt{mr}_2, \mathtt{slave}, \mathtt{ma}_2]\!] Y_1 \wedge [\![\mathtt{ma}_1]\!]$$
$$\nu Y_2.\langle\langle -\varepsilon\rangle\rangle \mathtt{tt} \wedge [\![\mathtt{mr}_1, \mathtt{mr}_2, \mathtt{slave}, \mathtt{ma}_1]\!] Y_2 \wedge [\![\mathtt{ma}_2]\!] X$$

- W cwb definiujemy:

$$FairEven \stackrel{\text{def}}{=} \mu X.\nu Y_1.\langle\langle -\varepsilon\rangle\rangle \mathtt{tt} \wedge [\![\mathtt{mr}_1, \mathtt{mr}_2, \mathtt{slave}, \mathtt{ma}_2]\!] Y_1 \wedge [\![\mathtt{ma}_1]\!]$$
$$\nu Y_2.\langle\langle -\varepsilon\rangle\rangle \mathtt{tt} \wedge [\![\mathtt{mr}_1, \mathtt{mr}_2, \mathtt{slave}, \mathtt{ma}_1]\!] Y_2 \wedge [\![\mathtt{ma}_2]\!] X$$
$$ValuesRead \stackrel{\text{def}}{=} Always([\![\mathtt{slave}]\!] FairEven)$$

- Ta własność dla *Msmie'* nie zachodzi - przynajmniej nie powinna:)

Obrazek pokazujący, że *ValuesRead* nie zachodzi



Poprawiony protokół

- Pomysł polega na wprowadzeniu czwartego bufora i dodatkowego stanu **oldmaster - o**.
- Kiedy procesor *k* wykonuje akcję $\mathbf{ma_k}$ mamy 3 przypadki:
  - Jeśli nie ma bufora **m** wtedy zmieniamy status bufora **n** na **m** i *k* czyta z tego bufora
  - Jeśli jest bufor b_1 o statusie **m**, i bufor b_2 o statusie **n** i nie ma bufora o statusie **o** wtedy b_1 zmieniamy na **o** i b_2 na **m** i *k* czyta z tego ostatniego
  - Jeśli jest bufor *b* o statusie **m** i albo jest bufor o statusie **o** albo nie ma bufora o statusie **n**, wtedy *k* jest dodawany do czytelników *b*

Slave

$$M'(\{(\mathbf{s},l),(\mathbf{n},l'),(\mathbf{i},l'')\} \cup B', r_o, r_m) \xrightarrow{\text{slave}} M'(\{(\mathbf{n},l),(\mathbf{s},l'),(\mathbf{i},l'')\} \cup B', r_o, r_m)$$

$$M'(\{(\mathbf{s},l),(\mathbf{n},l'),(\mathbf{i},l'')\} \cup B', r_o, r_m) \xrightarrow{\text{slave}} M'(\{(\mathbf{n},l),(\mathbf{i},l'),(\mathbf{s},l'')\} \cup B', r_o, r_m)$$

$$M'(\{(\mathbf{s},l),(\mathbf{i},l')\} \cup B', r_o, r_m) \xrightarrow{\text{slave}} M'(\{(\mathbf{n},l),(\mathbf{s},l')\} \cup B', r_o, r_m) \quad \forall l.(\mathbf{n},l) \notin B'$$

$$M'(\{(\mathbf{s},l),(\mathbf{n},l')\} \cup B', r_o, r_m) \xrightarrow{\text{slave}} M'(\{(\mathbf{n},l),(\mathbf{s},l')\} \cup B', r_o, r_m) \quad \forall l.(\mathbf{i},l) \notin B'$$

Master acquire

For $k \in MI$:

$$M'(\{(\mathrm{m}, l), (\mathrm{o}, l')\} \cup B', r_o, r_m) \xrightarrow{\mathrm{ma}_k} M'(\{(\mathrm{m}, l), (\mathrm{o}, l')\} \cup B', r_o, r_m \cup \{k\}) \quad k \notin (r_o \cup r_m)$$

$$M'(\{(\mathrm{m}, l)\} \cup B', r_o, r_m) \xrightarrow{\mathrm{ma}_k} M'(\{(\mathrm{m}, l)\} \cup B', r_o, r_m \cup \{k\}) \quad k \notin (r_o \cup r_m), \forall l.(\mathrm{n}, l) \notin B'$$

$$M'(\{(\mathrm{m}, l), (\mathrm{n}, l')\} \cup B', \emptyset, r_m) \xrightarrow{\mathrm{ma}'_k} M'(\{(\mathrm{o}, l), (\mathrm{m}, l')\} \cup B', r_m, \{k\}) \quad k \notin r_m$$

$$M'(\{(\mathrm{n}, l)\} \cup B', r_o, \emptyset) \xrightarrow{\mathrm{ma}'_k} M'(\{(\mathrm{m}, l)\} \cup B', r_o, \{k\}) \quad k \notin r_o$$

Master release

$$M'(B, r_o \cup \{k\}, r_m) \xrightarrow{\mathrm{mr}_k} M'(B, r_o, r_m) \qquad r_o \neq \emptyset, k \notin r_o$$

$$M'(\{(\mathbf{o}, l)\} \cup B', \{k\}, r_m) \xrightarrow{\mathrm{mr}_k} M'(\{(\mathbf{i}, l)\} \cup B', \emptyset, r_m)$$

$$M'(B, r_o, r_m \cup \{k\}) \xrightarrow{\mathrm{mr}_k} M'(B, r_o, r_m) \qquad r_m \neq \emptyset, k \notin r_m$$

$$M'(\{(\mathbf{m}, l), (\mathbf{n}, l')\} \cup B', r_o, \{k\}) \xrightarrow{\mathrm{mr}_k} M'(\{(\mathbf{i}, l), (\mathbf{n}, l')\} \cup B', r_o, \emptyset)$$

$$M'(\{(\mathbf{m}, l)\} \cup B', r_o, \{k\}) \xrightarrow{\mathrm{mr}_k} M'(\{(\mathbf{n}, l)\} \cup B', r_o, \emptyset) \qquad \forall l.(\mathbf{n}, l) \notin B'$$

Poprawiony protokół

$$MsmieNew \stackrel{\text{def}}{=} M'(\{(\mathrm{s},1),(\mathrm{i},2),(\mathrm{i},3),(\mathrm{i},4)\},\emptyset,\emptyset)$$