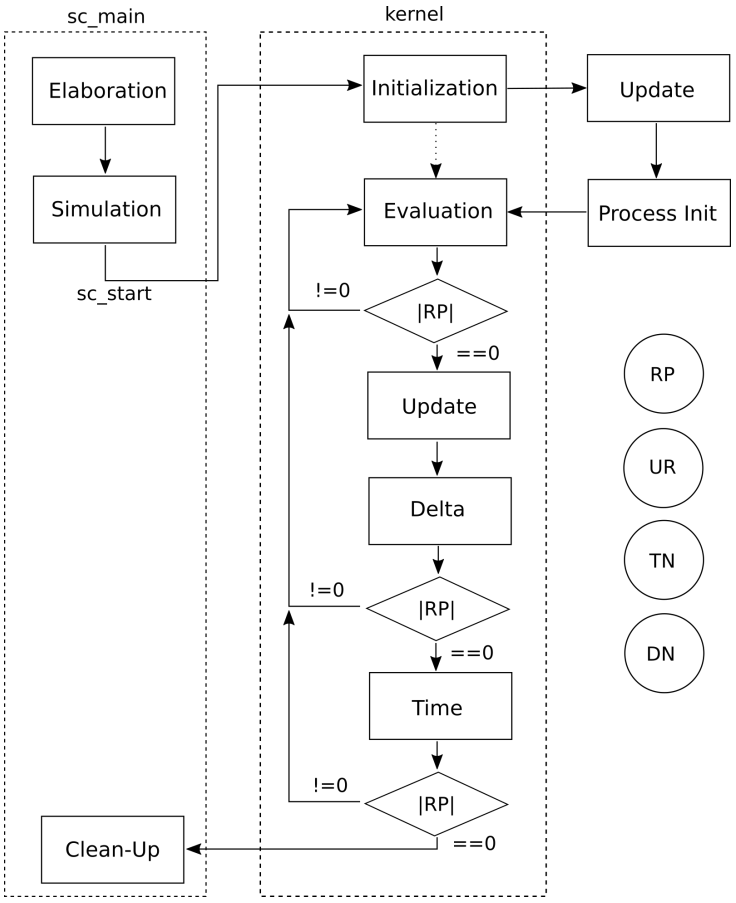
1.5 SystemC симулациони модел

Детаљан опис симулационог модела SystemC језика дат је у [2]. У наставку биће дато краће објашњење уз додатни пример.

Главни програм у SystemC симулацији пролази кроз три фазе (слика 1.3). Прва фаза је елаборација (Elaboration). У овој фази инстанцирају се све компоненте наведене у главном програму пре стартовања симулације. У другој фази симулирају се инстанциране компоненте (Simulation). У трећој фази се опционо уклањају неке компоненете и обрађују се резултати симулације.



Слика 1.3: SystemC симулациони кернел

Фаза симулације се започиње позивом функције `sc_start`. Састоји се од следећих фаза: иницијализација (Initialization), евалуација (Evaluation), ажурирање (Update), делта обавештења (Delta) и напредовање времена (Time). У току симеулације, кернел ажурира четири скупа:

скуп процеса спремних за покретање (RP, Runnable Processes), скуп захтева за ажурирање (UP, Update Request), скуп делта обавештења (DN - Delta Notification) и скуп временских обавештења (TN - Time Notification). Након стартовања симулационе фазе позива се иницијализација.

Током иницијализације позива се прво фаза ажурирања. Разлог позивања ове фазе биће објашњен у наредним поглављима. Након фазе ажурирања следи фаза иницијализације процеса. У овој фази, сви процеси који су регистровани током елаборације убацију у скуп RP. Прескачу се само процеси у којима је позвана функција `dont_initialize` након регистровања процеса. Након што су сви процеси убачени у скуп RP, фаза иницијализације се завршава и прелази се на фазу евалуације.

У току фазе евалуације, из скупа RP одабире се један процес. Одабрани процес избацује се из скуп RP и извршава се до тачке повратка или док се не наиђе на `wait` функцију или `suspend` методу. Проце се извршава без прекида, односно други процеси не могу да прекину извршавање покренутог процеса. Начин на који се процеси одабиру из скупа RP је зависан од имплементације.

У току свог извршавања процес можу послати обавештење (`sc_event - notify`). Постоје различити типови обавештења:

- Тренутно обавештење
- Делта обавештење
- Временско обавештење

Тренутно обавештење се постиже методом `notify sc_event` класе без аргумената, на пример `e.notify()`. Када се пошаље овај тип обавештења, сви процеси који чекају на догађај који је извршио методу `notify` се пребацују у скуп RP.

Делта обавештење постиже се позивом методе `notify`, класе `sc_event` са 0 временским интервалом као аргументом, `SC_ZERO_TIME`, на пример `e.notify(sc_core::SC_ZERO_TIME)`. Сва обавештења ове врсте стављају се у скуп DN и обрађују се током фазе делта обавештења.

Временско обавештење постиже се позивом методе `notify`, класе `sc_event` са не 0 времским интервалом, на пример `e.notify(5, sc_core::SC_NS)`. Сва обаветења ове врсте стављају се у скуп TN и обрађују се током фазе напредовања времена.

Када процес заврши своје извршавање одабире се нови процес из скупа RP па се он извршава и то се понавља све док скуп RP не буде празан. Када скуп PR постане празан прелази се на фазу ажурирања.

У току фазе ажурирања, узимају се сви захетви за ажурирање из скупа UR и обрађују се. У наредним поглављима биће више речи о овој фази и како се шаљу захтеви за ажурирање. Када се обраде сви захтеви за ажурирање, симулатор прелази на фазу делта обавештења.

У фази делта обавештења кернел симулатора проверава обавештења која се налазе у скупу DN. За свако обавештење из скупа кернел ради следеће:

- Одређује које инстанце процеса су осетљиве на ова обавештења
- Додаје све осетљиве процесе у скуп RP
- Уклања обрађено обавештење из скупа DN

Уколико је на крају ове фазе скуп RP празан, симулатор прелази у фазу напредовања времена. Уколико овај скуп није празан, симулатор се враћа на фазу евалуације.

У фази напредовања времена, кернел обрађује обавештења из скупа TN. Уколико постоје обавештења у овом скупу кернел симулатора одређује која временска обавештења су најближа тренутном времену симулатора. Симулатор затим помера време симулације до тог најближег времена. За свако од тих обавешетења симулатор обавља следеће:

- Одређује које инстанце процеса су осетљиве на ова обавештења
- Додаје све осетљиве процесе у скуп RP
- Уклања обрађено обавештење из скупа TN

Уколико је на крају ове фазе скуп RP празан фаза симулације се завршава. У супротном, кернел симулатора прелази на фазу евалуације.

Када се друга фаза SystemC симулације заврши, симулациони кернел се више не извршава. Симулација се наставља као обичан C++ програм. У овај фази се могу обрађивати резултати симулације, исписивати неке поруке или уконити динамички створене инстанце модула.

1.5.1 Пример симулације

Компонента `EventSys` садржи већи број догађаја и процеса (листинг 1.8). Процеси и догађају у овај компоненти смишљени су на погодан начин да на што једноставнији начин илуструју разне ситуације које су могуће унутар симулационог кернела.

Листинг 1.8: Пример тока симулације

```
#include <systemc>
#include <iostream>

SC_MODULE(EventSys)
{
public:
    sc_core::sc_event s0, s1, s2, s3, s4;

    SC_CTOR(EventSys)
    {
        SC_THREAD(proc0);
        SC_THREAD(proc1);
        SC_THREAD(proc2);
        SC_THREAD(proc3);
        SC_THREAD(proc4);
        dont_initialize();
        sensitive << s4;
    }

    void proc0();
    void proc1();
    void proc2();
    void proc3();
    void proc4();
    void print(int id);
    void hello(int id);
};

void EventSys::hello(int id)
{
    std::cout << "Hello from process " << id << " at " <<
                sc_core::sc_time_stamp() << std::endl;
}

void EventSys::print(int id)
{
    std::cout << "Process " << id << " at " <<
                sc_core::sc_time_stamp() << std::endl;
}

void EventSys::proc0()
{
    hello(0);
    wait(sc_core::SC_ZERO_TIME);
    s1.notify();
    while(true)
    {
        wait(s0);
        print(0);
        s1.notify(sc_core::SC_ZERO_TIME);
        s4.notify();
        s3.notify();
                wait(1, sc_core::SC_NS);
    }
}

void EventSys::proc1()
{
    hello(1);
    while(true)
    {
        wait(s1);
        print(1);
        s2.notify(1, sc_core::SC_NS);
    }
}

void EventSys::proc2()
{
    hello(2);
    while(true)
    {
        wait(s2);
        print(2);
        s0.notify(1, sc_core::SC_NS);
    }
}

void EventSys::proc3()
{
    hello(3);
```

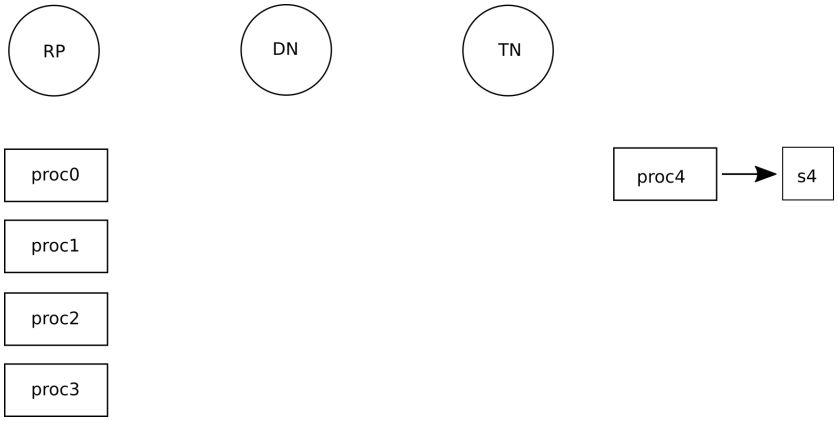
```
    while(true)
    {
        wait(s3);
        print(3);
    }
}

void EventSys::proc4()
{
    hello(4);
    while(true)
    {
        wait(s4);
        s4.notify();
        print(4);
    }
}

int sc_main(int argc, char* argv[])
{
    EventSys uut("UUT");

    sc_core::sc_start(10, sc_core::SC_NS);

    return 0;
}
```

Након елаборације инстанцирана је једна компонента класе `EventSys` и дато јој је име `UUT`. Након позива функције `sc_start`, кернел симулатора је прешао у фазу Initialize. Сва четири скупа од интереса су празна (слика 1.4). У подфази Update, у овом примеру, нема битних дешавања. У наставку разматрања неће бити обраћана пажња на скуп UN и фазу Update у главном току симулатора, пошто овај пример нема компоненти које користе обавештења за ажурирање.

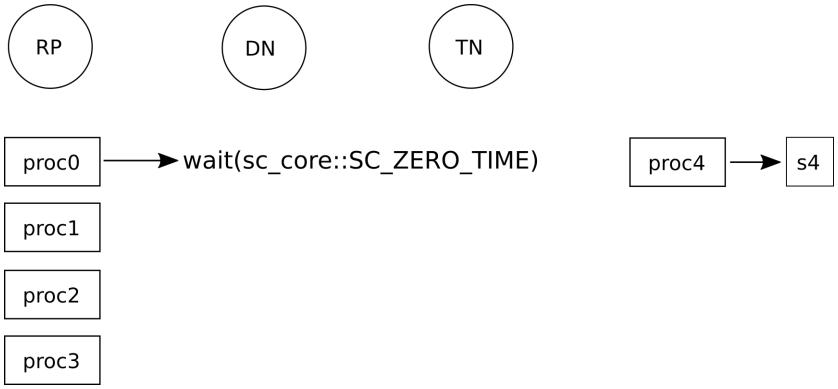
Слика 1.4: Корак 1

Компонента `EventSys` региструје неколико процеса: `proc0`, `proc1`, `proc2`, `proc3` и `proc4`. Процес `proc4`, након своје регистрације, проглашен је да се не иницијализује, помоћу `dont_initialize`. То значи да овај процес неће бити убачен у скуп RP након фазе Process Init. Сви остали процеси ће бити убачени у овај скуп након ове фазе (слика 1.5). Процес `proc4` биће у фази чекања након ове фазе и чекаће на догађај `s4` да би се активирао. Ппомоћу `sensitive « s4` означено је да је овај процес осетљив на догађај `s4`. Без тог означавања процес `proc4` никада се не би активирао, пошто није додат у скуп RP након фазе Process Init.

Након фазе Process Init, кернел се пребацује у фазу евалуације. Кернел бира један процес из скупа RP и извршава га. Начин бирања процеса није дефинисан SystemC стандардом и зависи од имплементације. Рецимо да је кернел одабрао `proc0` за извршавање у примеру (слика 1.6).

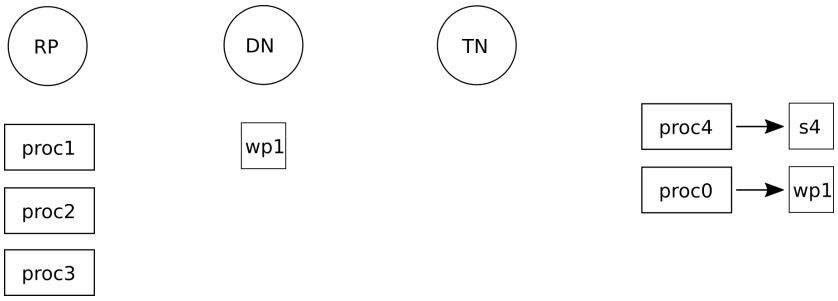


Слика 1.5: Корак 2



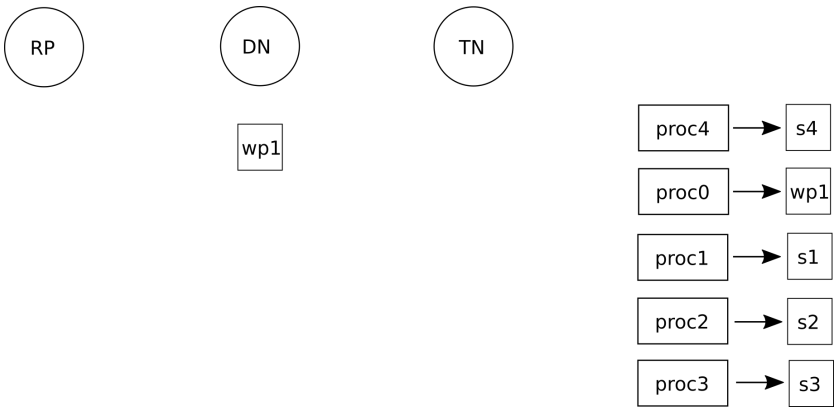
Слика 1.6: Корак 3

У току извршавања процеса `proc0`, штампа се порука `hello` методом, па се позива функција `wait` са параметром `SC_ZERO_TIME`. Овај позив направиће једно делта обавештење, зовимо га `wp1`, и процес `proc1` прећиће у стање чекања. Процес ће чекати да се обради `wp1` обавештење (слика 1.7).



Слика 1.7: Корак 4

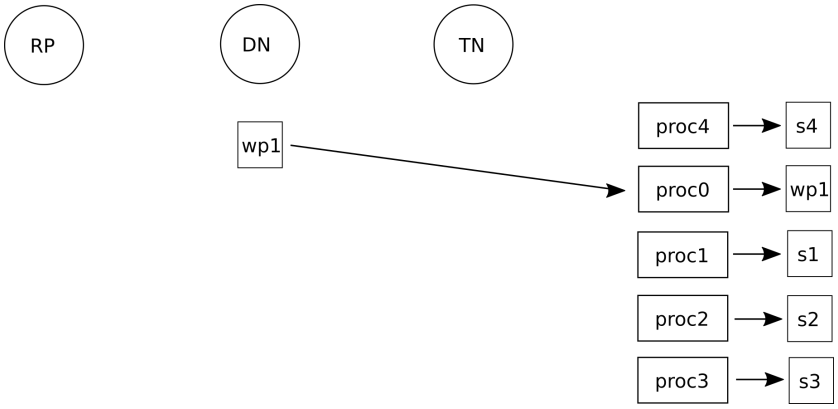
Након обраде прцеса proc1, сви остали процеси из скупа RP ће се извршити, неким редоследом. Сваки од њих ће одштампати своју поруку, методом hello и потом ће позвати функцију wait која ће као параметар добити догађај на који се чека. Сваки од процеса ће бити пребачен у стање чекања и остаће у њему док се не активира одговарајући догађај (слика 1.8). Сет RP ће бити празан, па ће се кернел пребацити у фазу обраде делта обавештења.



Слика 1.8: Корак 5

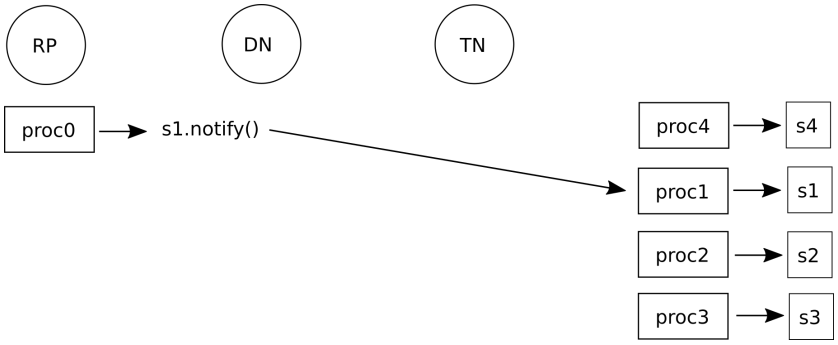
У фази делта обавештења, скуп DN има један елемент, `wp1` обавештење (слика 1.9). Кернел ће обрадити то обавештење, што значи да ће сви процеси који чекају на то обавештење бити пребачени у скуп RP. Само процес `proc1` чека на то обавештење, тако да ће бити пребачен

у скуп RP. Скуп DN је празан и скуп RP није празан, па се кернел пребацује у фазу евалуације.



Слика 1.9: Корак 6

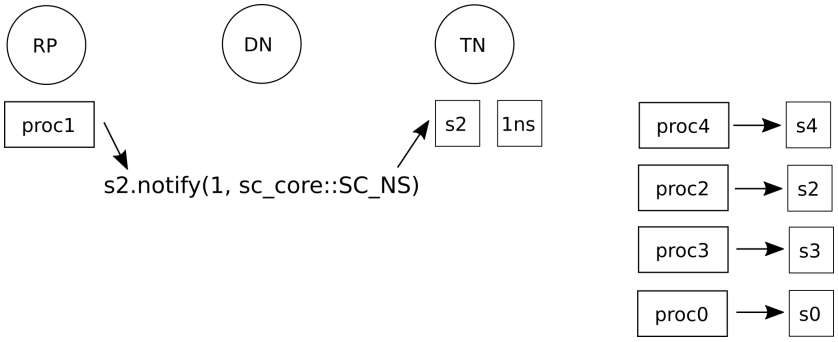
У фази евалуације се тренутно налази само један процес `proc0`, па се он извршава. У току његовог извршавања догађај `s1` шаље тренутно обавештење, `s1.notify()`. Процес `proc1`, који чека на ово обавештење, пребацује се одмах у скуп RP (слика 1.10). Процес `proc1` уклања се из скупа RP и пребацује у фазу чекања на обавештење од догађаја `s0`. Пошто скуп RP није празан, кернел симулатора остаје у фази евалуације.



Слика 1.10: Корак 7

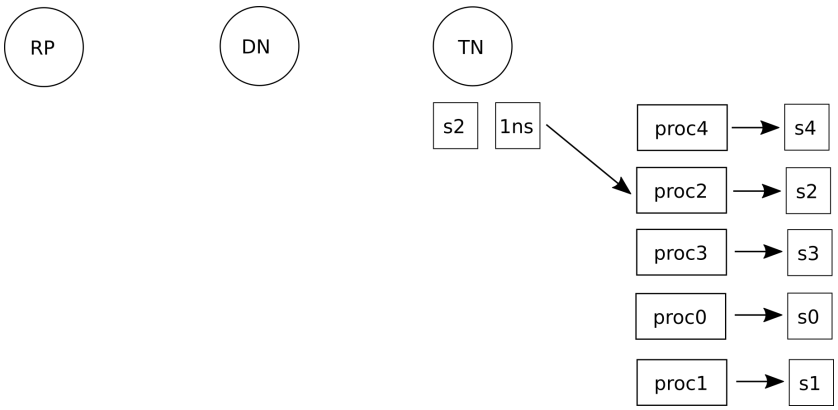
Скуп RP опет има један елемент, `proc1`, па се тај процес извршава. Током извршавања процеса, штампа се порука методом `print`, а после тога догађај `s2` шаље временско обавештење, `s2.notify(1, sc_core::SC_NS)` (слика 1.11). То обавештење се ставља у скуп TN. Након тога процес позива функцију `wait` и пребацује се у стање чекања. Након свега, скуп RP је празан и кернел се пребацује у фазу делта

обавештења. Скуп DN је исто празан, па се кернел одмах пребацује у фазу напредовања времена.



Слика 1.11: Корак 8

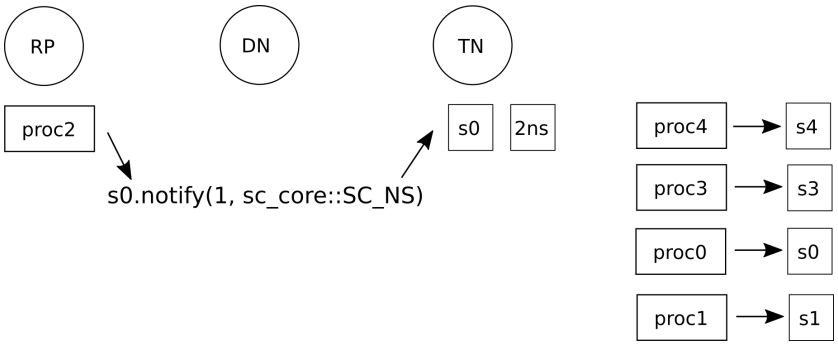
У фази напредовања времена, постоји само једно временско обавештење у скупу TN. То је обавештење догађаја `s1` са временском ознаком 2 ns (слика 1.12). Тек сада ће кернел по први пут повећати симулационо време од почетка симулације. Обавештење ће ставити процес `proc2` у скуп RP и обавештење ће се уклонити из скупа RN. Кернел ће се пребацити у фазу евалуације.



Слика 1.12: Корак 9

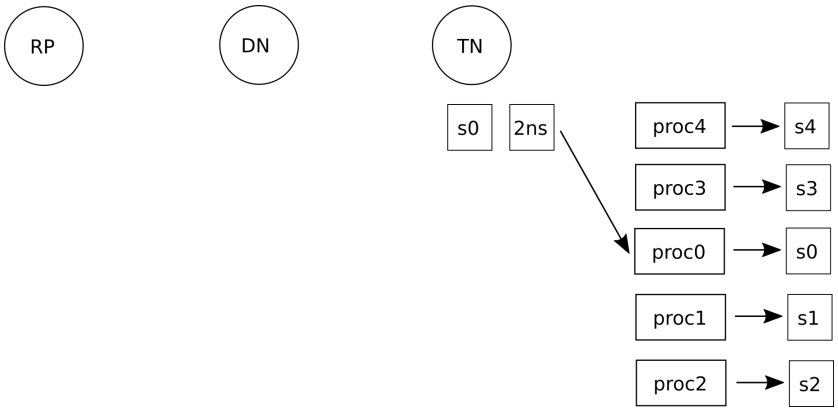
Процес `proc2` ће се извршити, пошто је једини у скупу, одштампаће своју поруку, па ће на крају догађај `s0` да пошаље своје временско обавештење `s0.notify(1,sc_core::SC_NS)` (слика 1.13). Временско ограничење биће означено временом од 2 ns, пошто је тренутно симулационо време 1 ns а на њега се додаје време обавештења које је исто 1 ns. Скуп RP ће бити празан, па ће се кернел пребацити у фазу

делта обавештења. Пошто је скуп DN празан, кернел ће се пребацити у фазу напретка времена.



Слика 1.13: Корак 10

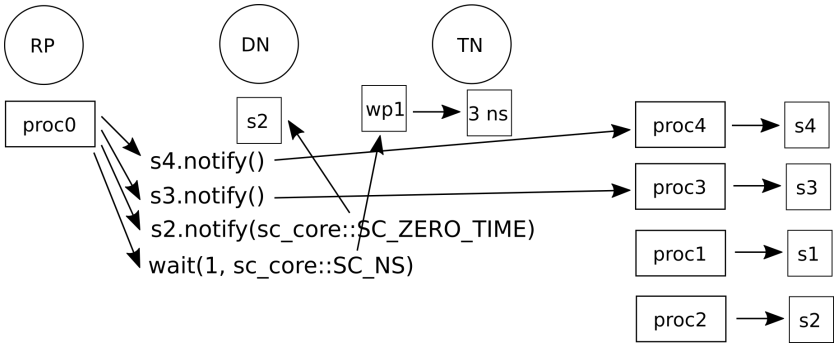
У фази напретка времена постоји само једно временско обавештење, које је од догађаја `s0` (слика 1.14) . Процес `proc0` биће пребачен у скуп RP, зато што чека на ово обавештење. Време симулације ће се повећати на 2 ns и кернел симулатора ће се пребацити у фазу евалуације.



Слика 1.14: Корак 11

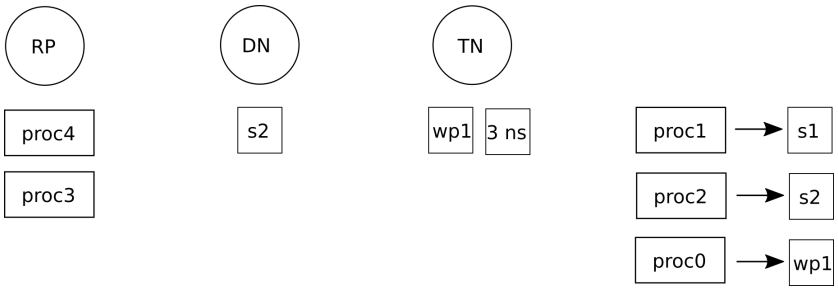
У том тренутку, само ће `proc0` бити у скупу RP. Процес `proc0` ће се извршити, одштампаће поруку, и направиће четири нова обавештења (слика 1.15). Прво, делта обавештење биће направљено позивом `s1.notify(sc_core::SC_ZERO_TIME)`. Затим ће бити направљена два тренутна обавештења позивима метода `s4.notify()` и `s3.notify()`. Последње обавештење биће направљено позивом `wait` функције.

На крају ове анализе, постојаће два процеса у скупу RP, стављена у скуп позивима тренутних обавештења (слика 1.16). Постојаће једно делта обавештење, са именом `wp1`, проузроковано позивом делта



Слика 1.15: Корак 12

обавештења. Уз то, биће и једно временско обавештење проузроковано позивом `wait` функције. На крају, биће и три процеса у стању чекања. Симулација ће се наставити од фазе евалуације, почевши од затеченог стања.



Слика 1.16: Коначно стање

1.5.2 Вежбе

1. Моделовати модул који садржи два процеса. Први процес ствара кружну секвенцу првих 64 Фибоначијевих бројева у случајним временским интервалима. Нови број се ствара у интервалу од 10 ns до 100 ns. Други процес исписује поруку 5 ns након што је број створен. За комуникацију међу процесима користити догађај. Порука треба да садржи број и тренутно време симулације.

2. Моделовати нуклеарни процес који се састоји од 1000 честица. Све честице моделовати истим модулом који ће се инстанцирати 1000 пута и садржаће један процес. Интеракцију међу честицама моделовати помоћу једног догађаја на који ће све честице бити осетљиве. Честица може бити у три стања: неактивно, активно,

експлодирано. Са вероватноћом од 1% честица ће постати активна уколико се активира догађај. Када честица постане активна она експлодира у случаном тренутку између 1 ns и 10 ns при чему активира исти догађај. Када честица експлодира није више осетљива на догађај. Исписати број честива које су експлодирале 100 ns након почетка симулације. У почетом тренутку, само једна честица је у активном стању, док су све остале у неактивном.