۶-۱- مدلسازی در B

در این بخش قصد داریم پروژه خود را در B مدلسازی کنیم؛ برای اینکار از دیاگرام‌های فعالیت و تکنیک‌هایی که برای بیان دقیق‌تر مدل‌های شئ‌گرا در فصل‌های قبل ارائه کردیم استفاده کرده‌ایم. قبل از اینکه لیست توصیف به زبان AMN را نشان دهیم قصد داریم ابتدا توضیح مختصری در مورد هر یک از بخشهای ماشین مجرد که کل سیستم را مدل می‌کند ارائه دهیم.

۶-۱-۱- ساختار استاتیک ماشین

در فهرست دستورات زیر که به زبان AMN نوشته شده است، ساختار استاتیک ماشین مجرد که کل سیستم را مدل می‌کند مشاهده می‌کنیم:

MACHINE
Entire (*max_pr*, *max_pg*, *root_server_as_sz*)
CONSTRAINTS
max_pr ∈ **NAT1** ∧
max_pg ∈ **NAT1** ∧
root_server_as_sz ∈ **NAT1** ∧
root_server_as_sz ≤ *max_pg*
INCLUDES
HostMachine
SEES
QueueType,
ProcessType,
AddressSpaceType,
MapletCodomainType,
NatPtrType,
NatASTupleType
CONSTANTS
root_server,
root_server_as,
running,
ready,
blocked
DEFINITIONS
INDIRECT_FRST == (λ(*pg*, *as*). (*pg* ∈$\mathbb{N}_1$∧ *as* ∈ *ADDRESS_SPACE* | *pg*));
INDIRECT_SCND == (λ(*pg*, *as*). (*pg* ∈$\mathbb{N}_1$∧ *as* ∈ *ADDRESS_SPACE* | *as*))
PROPERTIES
root_server ∈ (*PROCESS* - {*null_process*}) ∧
root_server_as ∈ (*ADDRESS_SPACE* - {*null_address_space*}) ∧
running ∈ (*QUEUE* - {*null_queue*}) ∧
ready ∈ (*QUEUE* - {*null_queue*}) ∧
blocked ∈ (*QUEUE* - {*null_queue*}) ∧
running ≠ *ready* ∧
running ≠ *blocked* ∧
ready ≠ *blocked*
VARIABLES
process,

process_pid,
process_parent_pid,
process_pager_pid,
process_exman_pid,
process_address_space,
process_waiting_ipc_from,
process_system_status,
queue,
queue_processes,
address_space,
address_space_size,
address_space_maplets,
maplet_codomain,
maplet_codomain_real,
maplet_codomain_indirect,
pool_of_free_page_directory_tables,
nat_ptr,
nat_ptr_val,
nat_as_tuple,
nat_as_tuple_val

همانطور که مشاهده می‌شود، نام این ماشین مجرد Entire می‌باشد که به علت اینکه کل سیستم را مدل می‌کند این نام را برایش انتخاب کرده‌ایم. این ماشین سه پارامتر با نام‌های max_pr، max_pg، و root_server_as_sz را تعریف می‌کند که به ترتیب حداکثر تعداد فرآیندهایی که می‌تواند در سیستم وجود داشته باشد، حداکثر تعداد صفحاتی که در فضای آدرس هر فرآیند می‌تواند جای بگیرد، و نهایتاً اندازه فضای آدرس فرآیند Root Server که بر حسب تعداد صفحات در این فضای آدرس بیان می‌شود. مشاهده می‌شود نوع و محدوده مقادیر مجاز برای این پارامترها را در بند **CONSTRAINTS** ماشین بیان کرده‌ایم.

بند **INCLUDES** این ماشین، حالات ماشینی را با نام HostMachine به مجموعه حالات ماشین Entire وارد می‌کند. ماشین HostMachine در حقیقت ماشین میزبانی را که سیستم عامل قرار است بر روی آن اجرا شود را مدل می‌کند؛ در ادامه در مورد این ماشین بحث خواهیم کرد. بند **SEES** تعداد ۵ عدد ماشین مجرد را به ماشین Entire معرفی می‌کند. هر یک از این ماشین‌ها مجموعه‌هایی نامتناهی را تعریف می‌کند که نشان طبق تعریف ۱ (تعریف کلاس‌ها)، مجموعه اسامی این مجموعه‌ها بیانگر مجموعه کلاس‌های مدل شئ‌گرا می‌باشد. این مجموعه‌ها نامتنهاهی بوده و هـــر عضو آنها شئ‌ای از آن کلاس خواهند بود. محتوای این مجموعه‌ها در این مرحله از توسعه سیستم مهم نیست از این رو این تعریف دقیق محتوای آن‌ها را به مراحل بعدی توسعه موکول کرده‌ایم. مجموعه QUEUE مجموعه شامـــل تمام اشیائی است که می‌توان از کلاس Queue نمونه‌سازی کرد، مجموعه

PROCESS بیانگر مجموعه تمام اشیائی است که می‌توان از کلاس Process نمونه‌سازی کرد، و به همین ترتیب مجموعه‌های ADDRESS_SPACE و MAPLET_CODOMAIN به ترتیب مجموعه‌هایی از تمامی اشیائی می‌باشند که می‌توان از کلاس‌های AddressSpace و MapletCodomain نمونه‌سازی کرد. هر یک از ماشین‌هایی را که ماشین Entire آنها را می‌بیند علاوه بر تعریف مجموعه تمام اشیاء کلاس‌ها، ثوابتی را نیز تعریف می‌کنند؛ این ثوابت در حقیقت اشیاء بخصوصی می‌باشند که نشان دهنده اشیاء تهی یا null از کلاس‌های تعریف شده هستند.

ثوابت سیستم را در بند **CONSTANTS** از ماشین تعریف کرده‌ایم. ثابت‌های root_server و root_server_as به ترتیب اشیائی از کلاس‌های Process و AddressSpace می‌باشند که شئ فرآیند و شئ فضای آدرس فرآیند Root Server را که همواره باید در سیستم حضور داشته باشند را مدل می‌کنند. ثابت‌های running، ready، و blocked اشیائی از نوع QUEUE هستند که به ترتیب سه صف در حال اجرا، آماده اجرا، و بلوک شده را نشان می‌دهند. انتخاب این اشیاء بصورت ثابت به این معنی نیست که حالت آن‌ها قابل تغییر نیست بلکه نشان دهنده این موضوع است که این اشیاء همواره در سیستم حضور دارند.

در بند **DEFINITION** از ماشین می‌توان ثوابتی را تعریف کرد که در برای ارائه توصیفات خیلی مرتب‌تر مناسب می‌باشند. در این بند برای عباراتی که بصورت تکراری در توصیف سیستم استفاده می‌شوند نام انتخاب می‌کنیم؛ ابزارهای B در فاز تحلیل لغوی توصیف هر وقوع این اسامی را با مقدار آن جایگذاری می‌کنند [CLS012]. در این بند با استفاده از زبان لمدا دو تابع تعریف کرده‌ایم که دارای دامنه یکسانی هستند ولی تابع INDIRECT_FRST مؤلفه اول زوج مرتب را بر می‌گرداند، حال آنکه تابع INDIRECT_SCND مؤلفه دوم زوج مرتب را به عنوان خروجی تولید می‌کند.

در بند **PROPERTIES** مقادیر و نوع ثوابت را تعریف کرده‌ایم. این بند بصورت یک گزاره منطقی بیان شده است، که نشان می‌دهد هر یک از ثوابت دارای چه نوعی هستند ولی مقدار دقیق این ثوابت در مراحل بعدی توسعه سیستم تعیین خواهد شد. نکته‌ایی لازم است ذکر کنیم این است که در این گذاره و عباراتی که در ادامه خواهیم دید مجموعه $\mathbb{N}_1$ به معنای زیر مجموعه‌ایی از اعداد طبیعی است که از یک شروع می‌شود.

بخش **VARIABLES** در یک ماشین با تعریف متغییرهای یـــک ماشین حالت سیستم را تعریف می‌کند. حالت یک ماشین بر حسب اینکه مقدار فعلی متغییرهای ماشین چیست تعیین می‌شود. در بین این متغییرها متناظر با هر نام کلاس یک متغییر با آن نام ولی با حروف کوچک تعریف شده است. این متغییرها تمام اشیائی را که در سیستم تا یک مقطع زمانی بخصوص از آن کلاس ایجاد شده است را در خود نگه می‌دارند. طبق تعریف ٦ این متغییرها باید زیر مجموعه‌ایی از oid آن کلاس باشند؛ برای مثال متغییر process تمام اشیائی را که تا کنون از نوع Process ایجاد شده‌اند را در خود نگه می‌دارند، بنابرین این متغییر باید زیر مجموعه‌ایی از PROCESS باشد (این محدودیت‌ها و سایر محدودیت‌ها از قبیل اندازه و حداقل و حداکثر مقدار هر یک از متغییرها را در ادامه و در بند **INVARIANT** ماشین تعریف خواهیم کرد). در حال حاضر فقط به اسامی و موارد استفاده هر یک از متغییرها بسنده می‌کنیم. در ادامه در جدول ١ لیست متغییرها و موارد استفاده هر یک را مشاهده خواهیم کرد.

جدول ١- متغییرهای سیستم به همراه توضیحات مختصری در مورد هر یک.

نام متغییر	توضیحات
process	مجموعه تمامی اشیائی از کلاس Process که تا این لحظه در سیستم ایجاد شده است. طبق تعریف ٦، این متغییر معادل با $\sigma_{CLASS}\ (Process)$ می‌باشد.
process_pid	تابعی می‌باشد که هر عضو مجموعه process را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقدار خصیصه pid هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}\ (Process)$.
process_parent_pid	تابعی می‌باشد که هر عضو مجموعه process را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقدار خصیصه parentPID هر شئ موجود در ســـیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}\ (Process)$.
process_pager_pid	تابعی می‌باشد که هر عضو مجموعه process را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقدار خصیصه

	pagerPID هر شئ موجود در ســـیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}(Process)$.
process_exman_pid	تابعی می‌باشد که هر عضو مجموعه process را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقــدار خصیصه exmanPID هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}(Process)$.
process_address_space	تابعی می‌باشد که هر عضو مجموعه process را به یک شئ از نوع AdressSpace نگاشت می‌کند. این تابع در اصل مقدار خصیصه address_space هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با $\sigma_{ATT}(Process)$ برای کلاس Process.
process_ipc_contents	تابعی می‌باشد که هر عضو مجموعه process را به یک شئ از نوع Message نگاشت می‌کند. این تابع در اصل مقدار خصیصه ipcContents هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}(Process)$.
process_waiting_ipc_from	تابعی می‌باشد که هر عضو مجموعه process را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقدار خصیصه waitingIPCFrom هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه $\sigma_{ATT}(Process)$.
queue_processes	تابعی می‌باشد که هر عضو مجموعه queue را به دنباله‌ایی از مجوعه‌ی process نگاشت می‌کند. این تابع در اصل مقدار خصیصه processes هر شئ موجود در سیستم را تعیین می‌کند (دقت کنید که تنها سه عـــدد شئ از نـــوع queue با نام‌های running، ready، و blocked وجود دارد و مجموعه queue بصورت ثابت تعریف شده است). طبق تعریف ٦، این متغییر

	معادل است با عضوی از مجموعه σ_{ATT} $(Queue)$.
address_space	مجموعه تمامی اشیائی از کلاس AdressSpace که تا این لحظه در سیستم ایجاد شده است. طبق تعریف ٦، این متغییر معادل با σ_{CLASS} $(AddressSpace)$ می‌باشد.
address_space_maplets	تابعی می‌باشد که هر عضو مجموعه address_space را به یک دنباله از نوع MapletCodomain نگاشت می‌کند (دنباله‌ایی از اشیاء موجود در مجموعه maplet_codomain). این تابع در اصل مقدار خصیصه maplets هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه σ_{ATT} $(AddressSpace)$.
maplet_codomain	مجموعه تمامی اشیائی از کلاس MapletCodmain که تا این لحظه در سیستم ایجاد شده است. طبق تعریف ٦، این متغییر معادل با عضوی از مجموعه σ_{CLASS} $(MapletCodomain)$ می‌باشد.
maplet_codomain_real	تابعی می‌باشد که هر عضو مجموعه maplet_codomain را به یک عدد طبیعی نگاشت می‌کند. این تابع در اصل مقدار خصیصه index هر شئ موجود در سیســـتم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه σ_{ATT} $(MapletCodomain)$.
maplet_codomain_indirect	تابعی می‌باشد که هر عضو مجموعه maplet_codomain را به یک دوتایی عدد طبیعی و شئ فضای آدرس نگاشت می‌کند. این تابع در اصل مقدار خصیصه indirect هر شئ موجود در سیستم را تعیین می‌کند. طبق تعریف ٦، این متغییر معادل است با عضوی از مجموعه σ_{ATT} $(MapletCodomain)$.

۶-۱-۲- گزاره تغییرناپذیر ماشین

پس از ارائه توضیح در مورد ساختار استاتیک ماشین لازم است در مورد گزاره تغییرناپذیر ماشین یا همان گزاره‌ایی که بصورت یک ترکیب وصلی در بند **INVARIANT** ماشین نوشته می‌شود توضیح

دهیم. همانطور که می‌دانیم و ذکر شد گزاره و شرط همیشه برقرار در سیستم بصورت یک ترکیب وصلی در ماشین نوشته می‌شود. در این گزاره علاوه بر تعیین نوع برای هر یک از متغییرهای ماشین، قواعد حاکم بر سیستم را نیز خواهیم نوشت. در مدل UML سیستم گزاره‌های تغییر ناپذیر سیستم را در مورد هر یک از کلاس‌ها که به زبان OCL نوشته شده بود را مشاهده کردیم. در این بخش با مراجعه به برخی از منابع ارائه کننده روش‌های ترجمه OCL به AMN مانند [LEDB01] و [LEDC01] و معناشناخت دلالتی زبـــان OCL می‌توان گزاره‌های تغییر ناپذیر سیستم را با حفظ معنا به AMN ترجمه کرد. در ادامه گزاره تغییر ناپذیر سیستم را می‌توانید مشاهده کنید در اینجا لازم است که توضیح مختصری در مورد هر یک از اجزای تشکیل دهنده این ترکیب وصلی ارائه دهیم.

سه جمله این گزاره نوع متغییرهای process، address_space، و maplet_codomain را با معرفی آن‌ها به عنوان زیرمجموعه‌هایی متناهی از مجموعه‌های PROCESS ، ADDRESS_SPACE، و MAPLET_CODOMAIN تعریـــف می‌کند. متغییر process_pid به عنوان تابعی یک به یک و کامل (یا Total) از مجموعه اشیاء نوع Process جاری به مجموعه اعداد طبیعی تعریف می‌شود، متغییرهای process_parent_pid، process_pager_pid، و process_exman_pid نیز به عنوان توابعی کامل از مجموعه اشیاء نوع Process جاری به مجموعه اعداد طبیعی تعریف می‌شوند. متغییر process_address_space به عنوان تابعی پوشا، کامل، و یک به یک (یا Bijective) از مجموعه اشیاء نوع Process جاری به مجموعه اشیاء نوع جاری نوع AddressSpace تعریف می‌شود؛ و به همین ترتیب نوع سایر متغییرها نیز تعریف می‌شود. نکته‌ایی که باید در مورد نوع متغییرها به آن دقت کنیم این است که برخی از قواعد حاکم بر سیستم نیز بطور خودکار اعمال می‌شوند، برای مثال یک به یک بودن متغییر process_pid تضمین می‌کند که خصیصه pid تمامی فرآیندهای موجود در سیستم یکتا خواهد بود و همچنین Bijective بودن متغییر process_address_space تضمین مـی‌کـنـد که کلاس‌های Process و AddressSpace با هم رابطه جزء-کل دارند. برد تابع queue_processes زیرمجموعه‌ایی از نوع **iseq** (process) می‌باشد که نشان دهنده این موضوع می‌باشد که در هر صف سیستم تکرار وجود ندارد. گزاره ۱۸ام بیان کننده این است که همواره باید فرآیند root_server در سیستم حضور داشته باشد. گزاره بعدی بیان می‌کند که همواره فرآیندی برای اجرا شدن وجود دارد. گزاره‌های بعدی نیز سایر قواعد حاکم بر سیستم را اعمال می‌کنند که نیازی به توضیح بیشتر احساس

نمی‌شود چرا که زبان AMN از نمادهای مرسوم در ریاضی و منطق استفاده می‌کند. سایر متغییرها مشابه آن چیزی هستند که توضیح دادیم، بنابراین از توضیح بیشتر می‌پرهیزیم.

INVARIANT

$queue \in \mathcal{F}(QUEUE) \wedge$
$process \in \mathcal{F}(PROCESS) \wedge$
$address\_space \in \mathcal{F}(ADDRESS\_SPACE) \wedge$
$maplet\_codomain \in \mathcal{F}(MAPLET\_CODOMAIN) \wedge$
$nat\_ptr \in \mathcal{F}(NAT\_PTR) \wedge$
$nat\_as\_tuple \in \mathcal{F}(NAT\_AS\_TUPLE) \wedge$
$queue = \{running, ready, blocked\} \wedge$
$null\_process \notin process \wedge$
$null\_address\_space \notin address\_space \wedge$
$null\_maplet\_codomain \notin maplet\_codomain \wedge$
$null\_nat\_ptr \notin nat\_ptr \wedge$
$null\_nat\_as\_tuple \notin nat\_as\_tuple \wedge$
$process\_pid \in process \rightarrowtail \mathbb{N}_1 \wedge$
$process\_parent\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_pager\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_exman\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_address\_space \in process \twoheadrightarrow\!\!\!\!\!\rightarrowtail address\_space \wedge$
$process\_waiting\_ipc\_from \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_system\_status \in process \rightarrow (REGS \rightarrow \mathbf{NAT}) \wedge$
$queue\_processes \in queue \rightarrow \mathbf{iseq}\ (process) \wedge$
$address\_space\_size \in address\_space \rightarrow \mathbb{N}_1 \wedge$
$address\_space\_maplets \in address\_space \rightarrow (\mathbf{seq1}\ (maplet\_codomain)) \wedge$
$maplet\_codomain\_real \in maplet\_codomain \rightarrow (nat\_ptr \cup \{null\_nat\_ptr\}) \wedge$
$maplet\_codomain\_indirect \in maplet\_codomain \rightarrow (nat\_as\_tuple \cup \{null\_nat\_as\_tuple\}) \wedge$
$nat\_ptr\_val \in nat\_ptr \rightarrow \mathbb{N} \wedge$
$nat\_as\_tuple\_val \in nat\_as\_tuple \rightarrow (\mathbb{N}_1 \times address\_space) \wedge$
$root\_server \in process \wedge$
$(\mathbf{size}\ (queue\_processes\ (running)) + \mathbf{size}\ (queue\_processes\ (ready))) > 0 \wedge$
$\mathbf{size}\ (queue\_processes\ (running)) \leq 1 \wedge$
$\cup\ qq.\ (qq \in queue \mid \mathbf{ran}\ (queue\_processes\ (qq))) = process \wedge$
$\cap\ qq.\ (qq \in queue \mid \mathbf{ran}\ (queue\_processes\ (qq))) = \varnothing \wedge$
$\forall as.\ (as \in address\_space \Rightarrow \mathbf{card}\ (address\_space\_maplets\ (as)) \leq max\_pg) \wedge$
$\forall ival.\ (ival : (\mathbf{ran}\ (maplet\_codomain\_indirect) - \{null\_nat\_as\_tuple\}) \Rightarrow$
$(INDIRECT\_FRST\ (nat\_as\_tuple\_val\ (ival)) \leq address\_space\_size$
$(INDIRECT\_SCND\ (nat\_as\_tuple\_val\ (ival))))) \wedge$
$\forall as\_obj.\ (as\_obj \in address\_space \Rightarrow \mathbf{size}\ (address\_space\_maplets\ (as\_obj))$
$= address\_space\_size\ (as\_obj)) \wedge$
$\forall as\_obj.\ (as\_obj \in address\_space \Rightarrow address\_space\_size\ (as\_obj) \leq max\_pg) \wedge$
$\mathbf{card}\ (process) \leq max\_pr \wedge$
$pool\_of\_free\_page\_directory\_tables \in \mathcal{F}(\mathbf{NAT}) \wedge$
$\mathbf{card}\ (pool\_of\_free\_page\_directory\_tables) + \mathbf{card}\ (process) \geq max\_pr \wedge$
$\forall ml.\ (ml \in maplet\_codomain \Rightarrow (maplet\_codomain\_real\ (ml) \neq null\_nat\_ptr \Rightarrow$
$maplet\_codomain\_indirect\ (ml) = null\_nat\_as\_tuple)) \wedge$
$\forall ml.\ (ml \in maplet\_codomain \Rightarrow (maplet\_codomain\_indirect\ (ml) \neq null\_nat\_as\_tuple \Rightarrow$
$maplet\_codomain\_real\ (ml) = null\_nat\_ptr))$

۶-۱-۳- مقدار دهی اولیه ماشین

هر ماشین مجردی که متغییرهایی برای آن تعریف شده است نیاز به یک عبارتی دارد که متغییرهای آن را مقداردهی اولیه می‌کند و آن را در یک حالت پیش‌فرض اولیه قرار می‌دهد. در زبان AMN این عبارت را در بند **INITIALISATION** از ماشین می‌نویسیم. با نوشتن هر عبارت مقدار دهی اولیه باید تضمین کنیم که نتایج حاصل از این عبارت از شرط مطلوبی که می‌خواهیم همواره در سیستم برقرار باشد تخطی نمی‌کند. در زیر عبارت مقداردهی اولیه سیستم را مشاهده می‌کنیم. البته این مقداردهی اولیه پایه‌ایی ترین حالت ممکن است و شاید در یک توسعه دیگر در سیستم نیازمند این باشیم که عبارات مقداردهی اولیه بسیار پیچیده‌تری را بنویسیم.

INITIALISATION
queue := {*running*, *ready*, *blocked*} ||
queue_processes := {*running* ↦ [*root_server*], *ready* ↦ [], *blocked* ↦ []} ||
process := {*root_server*} ||
address_space := {*root_server_as*} ||
ANY
mc, *np*
WHERE
mc ⊂ *MAPLET_CODOMAIN* ∧
np ⊂ *NAT_PTR* ∧
null_maplet_codomain ∉ *mc* ∧
null_nat_ptr ∉ *np* ∧
card (*mc*) = *root_server_as_sz* ∧
card (*np*) = *root_server_as_sz*
THEN
maplet_codomain := *mc* ||
nat_ptr := *np* ||
maplet_codomain_indirect := *mc* × {*null_nat_as_tuple*} ||
ANY
rm, *mp*, *npv*
WHERE
rm ∈ *mc* ⤖ *np* ∧
mp ∈ **perm** (*mc*) ∧
npv ∈ *np* ⤖ ℕ
THEN
maplet_codomain_real := *rm* ||
address_space_maplets := {*root_server_as* ↦ *mp*} ||
nat_ptr_val := *npv*
END
END ||
nat_as_tuple := ∅ ||
process_pid := {*root_server* ↦ 1} ||
process_parent_pid := {*root_server* ↦ 1} ||
process_pager_pid := {*root_server* ↦ 1} ||
process_exman_pid := {*root_server* ↦ 1} ||
process_address_space := {*root_server* ↦ *root_server_as*} ||
process_waiting_ipc_from := {*root_server* ↦ 1} ||
ANY
ss

WHERE
ss ∈ *REGS* → **NAT**
THEN
process_system_status := {*root_server* ↦ *ss*}
END ||
address_space_size := {*root_server_as* ↦ *root_server_as_sz*} ||
ANY
ff
WHERE
ff ∈ $\mathcal{F}$(**NAT**) ∧
card (*ff*) ≥ *max_pr*
THEN
pool_of_free_page_directory_tables := *ff*
END ||
nat_as_tuple_val := ∅

در عبارت مقداردهی اولیه سیستم که بصورت ترکیب موازی چند عبارت نوشته شده است، مشاهده می‌شود که در ابتدا یک فرآیند با نام root_server در سیستم در حال اجرا می‌باشد که pid آن برابر با عدد یک می‌باشد؛ برای سادگی فرض شده است که فضای آدرس این فرآیند تهی می‌باشد، حال آنکه در واقعیت و زمانی که قرار است این هسته بر روی یک سیستم کامپیوتری اجرا گردد فضای آدرس فرآیند root_server باید شامل کل حافظه فیزیکی سیستم مورد نظر باشد.

نکته‌ایی که باید در مورد اجرای هر عبارت AMN بدان توجه کرد، آثار جانبی می‌باشد که اجرای آن عبارت به جای می‌گذارد؛ طراح سیستم باید تضمین کند که اجرای هر یک از عباراتی که را توصیف می‌کنند از گزاره تغییرناپذیر سیستم تخطی نمی‌کند، به عبارتی دیگر اجرای عبارات مذکور سیستم را به یک حالت پایدار منتقل می‌کند. به اینگونه اثبات‌ها در مورد سیستم اثبات سازگاری و ثبات[1] سیستم گویند. در مورد یک ماشین مجرد علاوه بر عبارات AMN در مورد هر یک از بندهای ماشین نیز اثبات‌هایی انجام می‌شود، برای مثال طراح سیستم باید اثبات کند که مقادیری برای هر یک از ثوابت ماشین موجود است به شرطی که بند **PROPERTIES** ماشین صحیح باشد. خوشبختانه متد B بگونه‌ایی است که مهندس نرم‌افزار می‌تـواند در استفاده از آن از ابزارهای کامپیوتری استفاده کند. نرم‌افزار AtelierB از جمله ابزارهایی است که از استفاده از متد B پشتیبانی می‌کند؛ این نرم‌افزار با تولید تمامی زیراهدافی[2] که برای اثبات ثبات و درستی سیستم لازم است، درستی سیستم را تضمین می‌کند. این نرم‌افزار تعداد زیادی از قضایایی که اثبات آنها معادل با اثبات درستی سیستم هستند را بطور اتوماتیک و بر اساس مجموعه قواعدی که از قبل در داخل آن جاسازی شده است اثبات می‌کند.

[1] Machine Consistency.
[2] Sub-goal.

با وجود این همواره تعداد قضیه نیز اثبات نشده باقی می‌ماند که باید کاربر در اثبات آن‌ها به برنامه کمک کند. در ادامه فهرست کامل توصیف سیستم را در B خواهیم دید؛ اثبات درستی این توصیف را که با استفاده از نرم‌افزار AtelierB صورت گرفته است در پیوست ۱ آورده‌ایم. در بخش‌های بعدی این فصل نیز با نرم‌افزار AtelierB بیشتر آشنا خواهیم شد.

MACHINE

Entire (*max_pr*, *max_pg*, *root_server_as_sz*)

CONSTRAINTS

max_pr ∈ **NAT1** ∧
max_pg ∈ **NAT1** ∧
root_server_as_sz ∈ **NAT1** ∧
root_server_as_sz ≤ *max_pg*

INCLUDES

HostMachine

SEES

QueueType,
ProcessType,
AddressSpaceType,
MapletCodomainType,
NatPtrType,
NatASTupleType

CONSTANTS

root_server,
root_server_as,
running,
ready,
blocked

DEFINITIONS

INDIRECT_FRST == (λ(*pg*, *as*). (*pg* ∈ $\mathbb{N}_1$ ∧ *as* ∈ *ADDRESS_SPACE* | *pg*));
INDIRECT_SCND == (λ(*pg*, *as*). (*pg* ∈ $\mathbb{N}_1$ ∧ *as* ∈ *ADDRESS_SPACE* | *as*))

PROPERTIES

root_server ∈ (*PROCESS* - {*null_process*}) ∧
root_server_as ∈ (*ADDRESS_SPACE* - {*null_address_space*}) ∧
running ∈ (*QUEUE* - {*null_queue*}) ∧
ready ∈ (*QUEUE* - {*null_queue*}) ∧
blocked ∈ (*QUEUE* - {*null_queue*}) ∧
running ≠ *ready* ∧
running ≠ *blocked* ∧
ready ≠ *blocked*

VARIABLES

process,
process_pid,
process_parent_pid,
process_pager_pid,
process_exman_pid,
process_address_space,
process_waiting_ipc_from,
process_system_status,
queue,
queue_processes,
address_space,
address_space_size,
address_space_maplets,
maplet_codomain,
maplet_codomain_real,
maplet_codomain_indirect,
pool_of_free_page_directory_tables,
nat_ptr,
nat_ptr_val,
nat_as_tuple,
nat_as_tuple_val

INVARIANT

$queue \in \mathcal{F}(QUEUE) \wedge$
$process \in \mathcal{F}(PROCESS) \wedge$
$address\_space \in \mathcal{F}(ADDRESS\_SPACE) \wedge$
$maplet\_codomain \in \mathcal{F}(MAPLET\_CODOMAIN) \wedge$
$nat\_ptr \in \mathcal{F}(NAT\_PTR) \wedge$
$nat\_as\_tuple \in \mathcal{F}(NAT\_AS\_TUPLE) \wedge$
$queue = \{running, ready, blocked\} \wedge$
$null\_process \notin process \wedge$
$null\_address\_space \notin address\_space \wedge$
$null\_maplet\_codomain \notin maplet\_codomain \wedge$
$null\_nat\_ptr \notin nat\_ptr \wedge$
$null\_nat\_as\_tuple \notin nat\_as\_tuple \wedge$

$process\_pid \in process \rightarrowtail \mathbb{N}_1 \wedge$
$process\_parent\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_pager\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_exman\_pid \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_address\_space \in process \twoheadrightarrow address\_space \wedge$
$process\_waiting\_ipc\_from \in process \rightarrow \mathbb{N}_1 \wedge$
$process\_system\_status \in process \rightarrow (REGS \rightarrow \mathbf{NAT}) \wedge$
$queue\_processes \in queue \rightarrow \mathbf{iseq}\ (process) \wedge$
$address\_space\_size \in address\_space \rightarrow \mathbb{N}_1 \wedge$
$address\_space\_maplets \in address\_space \rightarrow (\mathbf{seq1}\ (maplet\_codomain)) \wedge$
$maplet\_codomain\_real \in maplet\_codomain \rightarrow (nat\_ptr \cup \{null\_nat\_ptr\}) \wedge$
$maplet\_codomain\_indirect \in maplet\_codomain \rightarrow (nat\_as\_tuple \cup \{null\_nat\_as\_tuple\}) \wedge$
$nat\_ptr\_val \in nat\_ptr \rightarrow \mathbb{N} \wedge$
$nat\_as\_tuple\_val \in nat\_as\_tuple \rightarrow (\mathbb{N}_1 \times address\_space) \wedge$
$root\_server \in process \wedge$
$(\mathbf{size}\ (queue\_processes\ (running)) + \mathbf{size}\ (queue\_processes\ (ready))) > 0 \wedge$
$\mathbf{size}\ (queue\_processes\ (running)) \leq 1 \wedge$
$\cup\ qq.\ (qq \in queue \mid \mathbf{ran}\ (queue\_processes\ (qq))) = process \wedge$
$\cap\ qq.\ (qq \in queue \mid \mathbf{ran}\ (queue\_processes\ (qq))) = \varnothing \wedge$
$\forall as.\ (as \in address\_space \Rightarrow \mathbf{card}\ (address\_space\_maplets\ (as)) \leq max\_pg) \wedge$
$\forall ival.\ (ival : (\mathbf{ran}\ (maplet\_codomain\_indirect) - \{null\_nat\_as\_tuple\}) \Rightarrow$
$(INDIRECT\_FRST\ (nat\_as\_tuple\_val\ (ival)) \leq address\_space\_size\ (INDIRECT\_SCND\ (nat\_as\_tuple\_val\ (ival))))) \wedge$
$\forall as\_obj.\ (as\_obj \in address\_space \Rightarrow \mathbf{size}\ (address\_space\_maplets\ (as\_obj)) = address\_space\_size\ (as\_obj)) \wedge$
$\forall as\_obj.\ (as\_obj \in address\_space \Rightarrow address\_space\_size\ (as\_obj) \leq max\_pg) \wedge$
$\mathbf{card}\ (process) \leq max\_pr \wedge$
$pool\_of\_free\_page\_directory\_tables \in \mathcal{F}(\mathbf{NAT}) \wedge$
$\mathbf{card}\ (pool\_of\_free\_page\_directory\_tables) + \mathbf{card}\ (process) \geq max\_pr \wedge$
$\forall ml.\ (ml \in maplet\_codomain \Rightarrow (maplet\_codomain\_real\ (ml) \neq null\_nat\_ptr \Rightarrow maplet\_codomain\_indirect\ (ml) = null\_nat\_as\_tuple)) \wedge$
$\forall ml.\ (ml \in maplet\_codomain \Rightarrow (maplet\_codomain\_indirect\ (ml) \neq null\_nat\_as\_tuple \Rightarrow maplet\_codomain\_real\ (ml) = null\_nat\_ptr))$

INITIALISATION

$queue := \{running, ready, blocked\} \parallel$

queue_processes := {*running* ↦ [*root_server*], *ready* ↦ [], *blocked* ↦ []} ||
process := {*root_server*} ||
address_space := {*root_server_as*} ||
ANY
    mc, *np*
WHERE
    mc ⊂ *MAPLET_CODOMAIN* ∧
    np ⊂ *NAT_PTR* ∧
    null_maplet_codomain ∉ *mc* ∧
    null_nat_ptr ∉ *np* ∧
    card (*mc*) = *root_server_as_sz* ∧
    card (*np*) = *root_server_as_sz*
THEN
    maplet_codomain := *mc* ||
    nat_ptr := *np* ||
    maplet_codomain_indirect := *mc* × {*null_nat_as_tuple*} ||
    ANY
        rm, *mp*, *npv*
    WHERE
        rm ∈ *mc* ⤔ *np* ∧
        mp ∈ **perm** (*mc*) ∧
        npv ∈ *np* ⤔ ℕ
    THEN
        maplet_codomain_real := *rm* ||
        address_space_maplets := {*root_server_as* ↦ *mp*} ||
        nat_ptr_val := *npv*
    END
END ||
nat_as_tuple := ∅ ||
process_pid := {*root_server* ↦ 1} ||
process_parent_pid := {*root_server* ↦ 1} ||
process_pager_pid := {*root_server* ↦ 1} ||
process_exman_pid := {*root_server* ↦ 1} ||
process_address_space := {*root_server* ↦ *root_server_as*} ||

process_waiting_ipc_from := {*root_server* ↦ 1} ||
ANY
ss
WHERE
ss ∈ *REGS* → **NAT**
THEN
process_system_status := {*root_server* ↦ *ss*}
END ||
address_space_size := {*root_server_as* ↦ *root_server_as_sz*} ||
ANY
ff
WHERE
ff ∈ $\mathcal{F}$(**NAT**) ∧
card (*ff*) ≥ *max_pr*
THEN
pool_of_free_page_directory_tables := *ff*
END ||
nat_as_tuple_val := ∅
OPERATIONS
abortProcess (*pid*) =
PRE
pid ∈ **NAT1** ∧
pid ∈ **ran** (*process_pid*) ∧
size (*queue_processes* (*running*)) > 0 ∧
card (*process*) > 0 ∧
pid ≠ *process_pid* (*root_server*)
THEN
LET
pp
BE
pp = *process_pid*$^{-1}$ (*pid*)
IN
LET
cp,

```
        bl
BE
        cp = first (queue_processes (running)) ∧
        bl = process_parent_pid^-1 [{pid}] ∪ process_pager_pid^-1 [{pid}] ∪ process_exman_pid^-1 [{pid}] ∪ {pp}
IN
        IF (cp = pp ∨ process_pid (cp) = process_parent_pid (pp) ∨ process_pid (cp) = process_pager_pid (pp) ∨
                process_pid (cp) = process_exman_pid (pp)) THEN
                IF (card ((ran (queue_processes (running)) ∪ ran (queue_processes (ready))) - bl) > 0) THEN
                        LET
                                pdt_to_be_free,
                                as_obj_to_be_del
                        BE
                                pdt_to_be_free = ∪ pr. (pr ∈ bl | {(process_system_status (pr)) (CR3)}) ∧
                                as_obj_to_be_del = process_address_space [bl]
                        IN
                                LET
                                        mpc_obj_to_be_del
                                BE
                                        mpc_obj_to_be_del = ∪ mpc_s. (mpc_s ∈ address_space_maplets [as_obj_to_be_del] | ran (mpc_s))
                                IN
                                        LET
                                                nat_ptr_to_be_free,
                                                nat_as_tuple_to_be_del
                                        BE
                                                nat_ptr_to_be_free = maplet_codomain_real [mpc_obj_to_be_del] - {null_nat_ptr} ∧
                                                nat_as_tuple_to_be_del = maplet_codomain_indirect [mpc_obj_to_be_del] –
                                                        {null_nat_as_tuple}
                                        IN
                                                maplet_codomain := maplet_codomain - mpc_obj_to_be_del ||
                                                ANY
                                                        root_server_phi_mpc,
                                                        nat_ptr_seq
                                                WHERE
                                                        root_server_phi_mpc ∈ perm
                                                                ((ran (address_space_maplets (root_server_as))
```

- **dom** (*maplet_codomain_real* ⩥ {*null_nat_ptr*}))
- **dom** (*maplet_codomain_indirect* ⩥ {*null_nat_as_tuple*})) ∧
nat_ptr_seq ∈ **perm** (*nat_ptr_to_be_free*)

THEN

LET

res_to_be_return

BE

res_to_be_return = **ran** (*root_server_phi_mpc* ⊗ *nat_ptr_seq*)

IN

maplet_codomain_real := (*mpc_obj_to_be_del* ⩤ *maplet_codomain_real*)
⩤ *res_to_be_return*

END

END ||
address_space := *address_space* - *as_obj_to_be_del* ||
address_space_size := *as_obj_to_be_del* ⩤ *address_space_size* ||
address_space_maplets := *as_obj_to_be_del* ⩤ *address_space_maplets* ||
process := *process* - *bl* ||
process_pid := *bl* ⩤ *process_pid* ||
process_parent_pid := *bl* ⩤ *process_parent_pid* ||
process_pager_pid := *bl* ⩤ *process_pager_pid* ||
process_exman_pid := *bl* ⩤ *process_exman_pid* ||
process_address_space := *bl* ⩤ *process_address_space* ||
process_waiting_ipc_from := *bl* ⩤ *process_waiting_ipc_from* ||
process_system_status := *bl* ⩤ *process_system_status* ||
pool_of_free_page_directory_tables := *pool_of_free_page_directory_tables* ∪
pdt_to_be_free ||

LET

err_tuples_to_be_del

BE

err_tuples_to_be_del = ∪ *tpl*. (*tpl* : (*nat_as_tuple* - *nat_as_tuple_to_be_del*) |
((λ*nt*. (*nt* ∈ *NAT_AS_TUPLE* ∧ *INDIRECT_SCND* (*nat_as_tuple_val* (*nt*))
∈ *as_obj_to_be_del* | {*nt*})) ∪
(λ*nt*. (*nt* ∈ *NAT_AS_TUPLE* ∧ *INDIRECT_SCND* (*nat_as_tuple_val* (*nt*))
∉ *as_obj_to_be_del* | ∅))) (*tpl*))

IN

nat_as_tuple := *nat_as_tuple* - (*nat_as_tuple_to_be_del* ∪ *err_tuples_to_be_del*) ||
nat_as_tuple_val := (*nat_as_tuple_to_be_del* ∪ *err_tuples_to_be_del*) ⩤
nat_as_tuple_val ||
LET
mpc_to_be_adjust
BE
mpc_to_be_adjust = *maplet_codomain_indirect*$^{-1}$ [*err_tuples_to_be_del*]
IN
maplet_codomain_indirect := (*mpc_obj_to_be_del* ⩤
maplet_codomain_indirect) ⩤
(*mpc_to_be_adjust*×{*null_nat_as_tuple*})
END
END ||
ANY
running_q,
ready_q,
blocked_q
WHERE
running_q ∈ **perm** (**ran** (*queue_processes* (*running*)) - *bl*) ∧
ready_q ∈ **perm** (**ran** (*queue_processes* (*ready*)) - *bl*) ∧
blocked_q ∈ **perm** (**ran** (*queue_processes* (*blocked*)) - *bl*)
THEN
queue_processes := *queue_processes* ⩤
{*ready* ↦ *ready_q*, *running* ↦ *running_q*, *blocked* ↦ *blocked_q*}
END
END
END
END
END
END
END
END;

pr_pid ← **createProcess** (*pagerPID, exmanPID, initialStatus, asSize*) =
PRE
pagerPID ∈ **NAT1** ∧
exmanPID ∈ **NAT1** ∧
initialStatus ∈ *REGS* → **NAT** ∧
asSize ∈ **NAT1** ∧
asSize ≤ *max_pg* ∧
pagerPID ∈ **ran** (*process_pid*) ∧
exmanPID ∈ **ran** (*process_pid*) ∧
size (*queue_processes* (*running*)) > 0 ∧
card (*process*) < *max_pr* ∧
card (*pool_of_free_page_directory_tables*) > 0
THEN
ANY
new_pr,
new_as,
new_pid,
parentPID,
new_maplets
WHERE
new_pr ∈ *PROCESS* - *process* ∧
new_as ∈ *ADDRESS_SPACE* - *address_space* ∧
new_pr ≠ *null_process* ∧
new_as ≠ *null_address_space* ∧
new_pid ∈$\mathbb{N}_1$ - **ran** (*process_pid*) ∧
parentPID ∈ *process_pid* [**ran** (*queue_processes* (*running*))] ∧
new_maplets ⊆ *MAPLET_CODOMAIN* - *maplet_codomain* ∧
card (*new_maplets*) = *asSize* ∧
null_maplet_codomain ∉ *new_maplets*
THEN
process := *process* ∪ {*new_pr*} ||
address_space := *address_space* ∪ {*new_as*} ||

process_pid (*new_pr*) := *new_pid* ||
process_parent_pid (*new_pr*) := *parentPID* ||
process_pager_pid (*new_pr*) := *pagerPID* ||
process_exman_pid (*new_pr*) := *exmanPID* ||
process_address_space (*new_pr*) := *new_as* ||
process_waiting_ipc_from (*new_pr*) := 1 ||
ANY
fpdt
WHERE
fpdt ∈ *pool_of_free_page_directory_tables*
THEN
process_system_status (*new_pr*) := *initialStatus* ⩤ {*CR3* ↦ *fpdt*} ||
pool_of_free_page_directory_tables := *pool_of_free_page_directory_tables* - {*fpdt*}
END ||
ANY
mapping
WHERE
mapping ∈ **perm** (*new_maplets*)
THEN
address_space_maplets (*new_as*) := *mapping* ||
address_space_size (*new_as*) := *asSize*
END ||
maplet_codomain := *maplet_codomain* ∪ *new_maplets* ||
maplet_codomain_real := *maplet_codomain_real* ∪ (*new_maplets* × {*null_nat_ptr*}) ||
maplet_codomain_indirect := *maplet_codomain_indirect* ∪ (*new_maplets* × {*null_nat_as_tuple*}) ||
queue_processes (*ready*) := *queue_processes* (*ready*) ← *new_pr* ||
pr_pid := *new_pid*
END
END;

forceSchedule (*pid*) =
PRE
pid ∈ **NAT1** ∧

pid ∈ **ran** (*process_pid*) ∧
size (*queue_processes* (*running*)) > 0
THEN
LET
pp,
cp,
cpid
BE
pp = *process_pid*$^{-1}$ (*pid*) ∧
cp = **first** (*queue_processes* (*running*)) ∧
cpid = *process_pid* (**first** (*queue_processes* (*running*)))
IN
IF (*cp* = *pp* ∨ *cpid* = *process_parent_pid* (*pp*) ∨ *cpid* = *process_pager_pid* (*pp*) ∨ *cpid* = *process_exman_pid* (*pp*)) **THEN**
IF *pp* ∈ **ran** (*queue_processes* (*blocked*)) **THEN**
ANY
blocked_q
WHERE
blocked_q ∈ **perm** (**ran** (*queue_processes* (*blocked*)) - {*pp*})
THEN
queue_processes := *queue_processes* ⩤ {*blocked* ↦ *blocked_q*, *ready* ↦ (*queue_processes* (*ready*) ← *pp*)}
END
END
END
END;

mapPage (*pageNo*, *pid*, *faultyPageNo*) =
PRE
pageNo ∈ **NAT1** ∧
pid ∈ **NAT1** ∧
faultyPageNo ∈ **NAT1** ∧
pid ∈ **ran** (*process_pid*) ∧
size (*queue_processes* (*running*)) > 0
THEN

```
LET
        pp,
        pp_as
BE
        pp = process_pid^-1 (pid) ∧
        pp_as = process_address_space (process_pid^-1 (pid))
IN
        IF pp ∈ ran (queue_processes (blocked)) THEN
                IF faultyPageNo ≤ address_space_size (pp_as) THEN
                        IF (maplet_codomain_real (address_space_maplets (pp_as) (faultyPageNo)) = null_nat_ptr) ∧
                                (maplet_codomain_indirect (address_space_maplets (pp_as) (faultyPageNo)) = null_nat_as_tuple) THEN
                                LET
                                        cas
                                BE
                                        cas = process_address_space (first (queue_processes (running)))
                                IN
                                        IF pageNo ≤ address_space_size (cas) THEN
                                                IF (maplet_codomain_real (address_space_maplets (cas) (pageNo)) ≠ null_nat_ptr) ∨
                                                        (maplet_codomain_indirect (address_space_maplets (cas) (pageNo)) ≠ null_nat_as_tuple)
                                                THEN
                                                        IF (maplet_codomain_real (address_space_maplets (cas) (pageNo)) ≠ null_nat_ptr) THEN
                                                                ANY
                                                                        new_nat_as_tuple
                                                                WHERE
                                                                        new_nat_as_tuple ∈ NAT_AS_TUPLE - nat_as_tuple ∧
                                                                        new_nat_as_tuple ≠ null_nat_as_tuple
                                                                THEN
                                                                        nat_as_tuple := nat_as_tuple ∪ {new_nat_as_tuple} ||
                                                                        nat_as_tuple_val (new_nat_as_tuple) := (pageNo ↦ cas) ||
                                                                        maplet_codomain_indirect (address_space_maplets (pp_as)
                                                                                (faultyPageNo)) := new_nat_as_tuple
                                                                END
                                                        ELSE
```

maplet_codomain_indirect (*address_space_maplets* (*pp_as*) (*faultyPageNo*)) :=
maplet_codomain_indirect (*address_space_maplets* (*cas*) (*pageNo*))

END
END
END
END
END
END
END
END;

grantPage (*pageNo*, *pid*, *faultyPageNo*) =
PRE
pageNo ∈ **NAT1** ∧
pid ∈ **NAT1** ∧
faultyPageNo ∈ **NAT1** ∧
pid ∈ **ran** (*process_pid*) ∧
pid ∉ *process_pid* [**ran** (*queue_processes* (*running*))] ∧
size (*queue_processes* (*running*)) > 0
THEN
LET
pp,
pp_as
BE
pp = *process_pid*$^{-1}$ (*pid*) ∧
pp_as = *process_address_space* (*process_pid*$^{-1}$ (*pid*))
IN
IF *pp* ∈ **ran** (*queue_processes* (*blocked*)) **THEN**
IF *faultyPageNo* ≤ *address_space_size* (*pp_as*) **THEN**
IF (*maplet_codomain_real* (*address_space_maplets* (*pp_as*) (*faultyPageNo*)) = *null_nat_ptr*) ∧
(*maplet_codomain_indirect* (*address_space_maplets* (*pp_as*) (*faultyPageNo*)) = *null_nat_as_tuple*) **THEN**
LET
cas

```
                BE
                        cas = process_address_space (first (queue_processes (running)))
                IN
                        IF pageNo ≤ address_space_size (cas) THEN
                                IF (maplet_codomain_real (address_space_maplets (cas) (pageNo)) ≠ null_nat_ptr) ∨
                                        (maplet_codomain_indirect (address_space_maplets (cas) (pageNo)) ≠ null_nat_as_tuple)
                                THEN
                                        IF (maplet_codomain_real (address_space_maplets (cas) (pageNo)) ≠ null_nat_ptr) THEN
                                                LET
                                                        cas_real
                                                BE
                                                        cas_real = maplet_codomain_real (address_space_maplets (cas) (pageNo))
                                                IN
                                                        maplet_codomain_real := maplet_codomain_real ⩤
                                                                {address_space_maplets (cas) (pageNo) ↦ null_nat_ptr,
                                                                    address_space_maplets (pp_as) (faultyPageNo) ↦ cas_real}
                                                END
                                        ELSE
                                                LET
                                                        cas_ind
                                                BE
                                                        cas_ind = maplet_codomain_indirect (address_space_maplets (cas)
                                                                (pageNo))
                                                IN
                                                        maplet_codomain_indirect := maplet_codomain_indirect ⩤
                                                                {address_space_maplets (pp_as) (faultyPageNo) ↦ cas_ind,
                                                                        address_space_maplets (cas) (pageNo) ↦
                                                                                null_nat_as_tuple}
                                                END
                                        END
                                END
                        END
                END
        END
```

```
                    END
                END
            END
END;

reclaimPage (pageNo) =
PRE
        pageNo ∈ NAT1 ∧
        size (queue_processes (running)) > 0
THEN
        LET
                cas
        BE
                cas = process_address_space (first (queue_processes (running)))
        IN
                LET
                        ind_bl
                BE
                        ind_bl = nat_as_tuple_val^-1 [{pageNo ↦ cas}]
                IN
                        nat_as_tuple := nat_as_tuple - ind_bl ||
                        nat_as_tuple_val := ind_bl ⩤ nat_as_tuple_val ||
                        LET
                                ml_to_be_null
                        BE
                                ml_to_be_null = maplet_codomain_indirect^-1 [ind_bl]
                        IN
                                maplet_codomain_indirect := maplet_codomain_indirect ⩤ (ml_to_be_null × {null_nat_as_tuple})
                        END
                END
        END
END;

schedule =
PRE
```

ran (*status*) ⊆ **NAT**
THEN
IF *queue_processes* (*running*) = [] **THEN**
queue_processes := *queue_processes* ⩤ {*running* ↦ [**first** (*queue_processes* (*ready*))],
ready ↦ **tail** (*queue_processes* (*ready*))} ||
storeMachineStatus (*process_system_status* (**first** (*queue_processes* (*ready*))))
ELSE
IF *queue_processes* (*ready*) ≠ [] **THEN**
process_system_status (**first** (*queue_processes* (*running*))) := *status* ||
queue_processes := *queue_processes* ⩤ {*ready* ↦ **tail** (*queue_processes* (*ready*) ← **first** (*queue_processes* (*running*))),
running ↦ [**first** (*queue_processes* (*ready*))]} ||
storeMachineStatus (*process_system_status* (**first** (*queue_processes* (*ready*))))
END
END
END;

sendMessage (*pid*) =
PRE
pid ∈ **NAT1** ∧
pid ∈ **ran** (*process_pid*) ∧
size (*queue_processes* (*running*)) > 0
THEN
LET
pp
BE
pp = *process_pid*$^{-1}$ (*pid*)
IN
IF *pp* ∈ **ran** (*queue_processes* (*blocked*)) **THEN**
ANY
current_pid,
current_proc
WHERE
current_pid ∈ *process_pid* [**ran** (*queue_processes* (*running*))] ∧

```
                    current_proc ∈ ran (queue_processes (running))
                THEN
                    IF process_waiting_ipc_from (pp) = current_pid THEN
                        process_system_status (pp) := process_system_status (pp) <+
                            ({EAX, EBX, ECX, EDX, ESI, EDI} <| process_system_status (current_proc)) ||
                        ANY
                            blocked_q
                        WHERE
                            blocked_q ∈ perm (ran (queue_processes (blocked)) - {pp})
                        THEN
                            queue_processes := queue_processes <+ {ready ↦ (queue_processes (ready) ← pp),
                                blocked ↦ blocked_q}
                        END
                    END
                END
            END
        END
END;

receiveMessage (pid) =
PRE
    pid ∈ NAT1 ∧
    size (queue_processes (running)) > 0 ∧
    size (queue_processes (ready)) > 0
THEN
    LET
        current
    BE
        current = first (queue_processes (running))
    IN
        queue_processes := queue_processes <+ {running ↦ [], blocked ↦ (queue_processes (blocked) ← current)} ||
        process_waiting_ipc_from := process_waiting_ipc_from <+ {current ↦ pid}
    END
END;
```

dispatch = **BEGIN skip END**;

pid ← **getPID** =
PRE
    size (*queue_processes* (*running*)) > 0
THEN
    LET
        cp
    BE
        cp = **first** (*queue_processes* (*running*))
    IN
        pid := {1 ↦ *process_pid* (*cp*), 2 ↦ *process_parent_pid* (*cp*), 3 ↦ *process_pager_pid* (*cp*), 4 ↦ *process_exman_pid* (*cp*)}
    END
END

END

۶-۲- نرم‌افزار AtelierB ابزاری برای اثبات درستی

همانطور که در انتهای بخش ذکر شد اثبات درستی سیستم را با استفاده از نرم‌افزار AtelierB انجام داده‌ایم. متن کامل اثبات‌ها را در پیوست ۱ آورده‌ایم؛ در این بخش قصد داریم این ابزار را معرفی کرده و میزان کمک این ابزار را در پیشبرد توسعه سیستم ذکر کنیم.

۶-۲-۱- معرفی نرم‌افزار AtelierB

نرم‌افزار AtelierB ابزاری است که از متد B حمایت کرده، و می‌توان از آن برای اثبات درستی سیستم استفاده کنیم. این نرم‌افزار توسط شرکت فرانسوی ClearSy توسعه داده شده است. از امکانات این نرم‌افزار می‌توان به ویرایش‌گر قدرتمند برای تایپ کردن توصیف‌ها B، ابزار برای Type Check کردن، ابزار برای تولید وظایف (Proof Obligation)، و نهایتاً دستیار اثبات اشاره کرد. نسخه چهارم این نرم‌افزار به همراه تمامی مستنداتش رایگان بوده، و می‌توان از طریق شبکه اینترنت[۳] به آن دسترسی پیدا کرد. لازم به ذکر است که تحت اکثر سیستم‌عامل‌های مرسوم قابل اجرا است.

۶-۲-۲- میزان کمک کمک نرم‌افزار AtelierB در تسریع توسعه سیستم

همانطور که ذکر شد ابزار AtelierB قادر به تولید خودکار وظایف اثبات (یا Proof Obligation) و نیز اثبات خودکار این وظایف می‌باشد. با وجود اینکه ابزار قادر به اثبات بیش از نیمی از وظایف است، بسیاری از وظایف اثبات نشده باقی می‌مانند و مهندس نرم‌افزار باید با استفاده از اثبات کننده محاوره‌ایی سیستم اقدام به اثبات این وظایف کند. در ادامه دو جدول را خواهیم دید که در آن‌ها نام ماشین‌هایی است که برای توسعه سیستم طراحی شده‌اند. مقابل هر یک از این ماشین‌ها تعداد کل وظایف و تعدادی را که سیستم توانسته است بصورت اتوماتیک آن‌ها را اثبات کند آورده شده است. اولین جدول مربوط به حالت سیستم قبل از استفاده از اثبات کننده اتوماتیک است و جدول بعدی هم مربوط به حالت سیستم بعد از استفاده از اثبات کننده اتوماتیک می‌باشد.

اثبات بیش از نیمی از وظایف توسط ماشین کار توسعه سیستم را بسیار آسان می‌کند. علاوه بر این در بسیاری از مقالات ذکر شده است اثبات‌های ماشینی بسیار ارزشمندتر از اثبات‌های دستی می‌باشند [DRDB10]؛ این به دلیل کاهش خطا در روند اثبات‌های ماشینی است.

[3] http://www.atelierb.eu/

Component	TC	POG	nPO	nUN	%Pr	B0C
AddressSpaceType	OK	OK	0	0	–	–
Entire	OK	OK	565	565	0	–
HostMachine	OK	OK	1	1	0	–
MapletCodomainType	OK	OK	0	0	–	–
NatASTupleType	OK	OK	0	0	–	–
NatPtrType	OK	OK	0	0	–	–
ProcessType	OK	OK	0	0	–	–
QueueType	OK	OK	0	0	–	–

جدول ۲- وضعیت وظایف اثبات قبل از استفاده از ابزاراثبات خودکار AtelierB.

Component	TC	POG	nPO	nUN	%Pr	B0C
AddressSpaceType	OK	OK	0	0	–	–
Entire	OK	OK	565	238	57	–
HostMachine	OK	OK	1	1	0	–
MapletCodomainType	OK	OK	0	0	–	–
NatASTupleType	OK	OK	0	0	–	–
NatPtrType	OK	OK	0	0	–	–
ProcessType	OK	OK	0	0	–	–
QueueType	OK	OK	0	0	–	–

جدول ۳- وضعیت وظایف اثبات بعد از استفاده از ابزار اثبات خودکار AtelierB.

۶-۳- نتیجه‌گیری

در این فصل با استفاده از چهارچوبی که برای ترجمه در فصول قبل ذکر شده بود، مدل‌های UML سیستم را به مدل B ترجمه کردیم. برای اینکه اثبات درستی سیستم بیشتر قابل اعتماد باشد از ابزار اثبات AtelierB برای اثبات درستی توصیف استفاده کردیم. این ابزار با استفاده از اثبات کننده خودکار می‌تواند زمان لازم را برای یک توسعه B به حداقل برساند.

ممکن است متوجه این موضوع شده باشید که در توصیف B ارائه شده در این فصل برخی از ماشین‌هایی را که در توسعه از آن‌ها استفاده کرده‌ایم را توصیف نکردیم. توصیف این ماشین را می‌توانید در پیوست ۲ بیابید. در ادامه راهکارهایی را برای پالایش و پیاده‌سازی سیستم ارائه خواهیم داد.