۷-۱- پیش بسوی پالایش و پیاده‌سازی

در این فصل قصد داریم اقدامات لازم جهت پیاده سازی مدل توسعه داده شده را شرح دهیم. از آنجایی که پیاده‌سازی کامل این سیستم در یک زبان برنامه‌سازی مستلزم صرف وقت بیشتر از یک سال و چند ماه است و نیز نیازمند این است که توسط تیمی بزرگتر از یک تیم تک نفری انجام شود، در این پروژه صرفاً شیوه پالایش و آماده‌سازی آن برای پیاده‌سازی را بیان خواهیم کرد.

۷-۱-۱- حذف اطلاعات اضافی

در منـابع مختلف از جمله [SCHN01] گفته می‌شود که می‌توان در هر مرحله از پالایش توصیف‌های خود تعدادی از متغییرهای تعریف کننده فضای حالت ماشین مورد نظر را حذف و یا اینکه نوع آن‌ها را تغییر دهیم. در برخی نیز عدم قطعیت و گاهی مکان عدم قطعیت حذف و یا تغییر پیدا می‌کند. در مورد این پروژه برخی از متغییرهای ماشین اضافی بودند و در اصل نباید در نظر گرفته می‌شد، برای مثال نگهداری تمامی اشیاء ایجاد شده در سیستم به هیچ وجه نیاز نخواهد بود و این کار صرفاً به خاطر روش ترجمه‌ایی بود که ارائه داده بودیم.

در این پروژه در مراحل بعدی پالایش اطلاعات زمان اجرای سیستم را به بیرون از توصیف اصلی منتقل کردیم، بدین ترتیب ایجاد و حذف اشیاء را صرفاً با استفاده از دو عملگر ساخت شئ و تخریب آن انجام می‌دهیم. برای مثال در مورد اشیاء Process که قبلاً تماماً توسط ماشین Entire حذف و ایجاد می‌شد، در یک مرحله از پالایش به بیرون از این ماشین منتقل شده و اشیاء Process توسط دو عملگر constructProcessObject و destructProcessObject به ترتیب ایجاد و حذف شدند. با این کار ماشین Process مثل کلاسی شد که می‌توان اشیائی از آن را نمونه سازی و یا اشیائی از این نوع را حذف کرد. در ادامه توصیف تمامی این ماشین‌ها را خواهیم دید.

دسترسی به خصائص اشیاء ایجاد شده نیز از طریق عملگرهای ماشین‌های مدیریت کننده اشیاء صورت می‌پذیرد. برای مثال از طریق عملگر getPID می‌توان به خصیصه PID یک شئ از نوع Process دسترسی داشته باشیم. ولی این اعمال ساده بوده و همواره به مقداردهی و یا خواندن یک متغییر محدود می‌شوند.

۷-۱-۲- حذف عدم قطعیت

طبق [SCHN01] در هر مرحله از پالایش می‌توان عدم قطعیت را حذف و یا تغییر مکان[۱] داد. در این پروژه هم در جاهای مختلف ماشین Entire که توصیف مجرد اصلی کل سیستم را در بر دارد از عدم قطعیت استفاده کرده‌ایم. در مراحل بعدی پالایش برخی از این توصیف‌های غیر قطعی به ماشین‌های مدیریت اشیاء منتقل شدند (تغییر مکان) و برخی نیز با استفاده از توصیفات واقعی[۲] حذف شدند (حذف).

۷-۱-۳- شیوه پالایش و پیاده‌سازی

روش ترجمه ما بگونه‌ایی بود که بیش از یک مرحله پالایش برای توصیف نیاز نشد، به همین خاطر بعد از توصیف مجرد در Entire بطور مستقیم اقدام به ساخت یک ماشین پیاده‌سازی (یا IMPLEMENTATION) با نام EntireI کردیم. متاسفانه ماشین پیاده‌سازی را نتوانستیم بطور کامل توصیف و صحت سنجی کنیم. ماشین EntireI ماشین‌هایی را وارد (یا IMPORT) می‌کند، این ماشین‌ها همان ماشین‌های مدیریت زمان اجرای اشیاء هستند. نکته‌ایی که در اینجا باید ذکر کنیم این است که ماشین‌های مدیریت اشیاء نباید پالایش شوند چرا که در هیچ زبان برنامه نویسی شئ‌گرایی برنامه‌نویس مسئول حسابداری تمام اشیاء ایجاد شده در سیستم نیست، به همین خاطر در پروژه خود هیچ یک از ماشین‌های مدیریت اشیاء را پالایش و نکردیم و آن‌ها را بطور مستقیم بصورت کلاس‌هایی در زبان مقصد (C++) پیاده‌سازی کردیم که ایجاد و حذف اشیاء آن‌ها توسط عملگرهای new و delete[۳] صورت می‌گیرد. کلاس‌های پیاده‌سازی شده باید دارای عملگرهایی برای دسترسی به اعضای داده‌ایی کلاس‌ها داشته باشند، لازم به ذکر است که از آنجایی که این عملگرها با استفاده از زبان OCL بصورت دقیق بیان شده‌اند و همچنین از آنجایی که این عملگرها عملکر بسیار ساده‌ایی دارند نیازی به صحت سنجی آن‌ها وجود ندارد (علی رغم سادگی اثبات سازگاری این ماشین‌ها) با اطمینان بیشتری می‌توانیم عمل پیاده‌سازی را انجام دهیم. در زیر نمونه‌ایی از ماشین مدیریت اشیاء نوع PROCESS و نیز پیاده‌سازی آن‌ها در C++ را مشاهده می‌کنید.

---

[۱] منظور از تغییر مکان، تغییر شیوه ایجاد عدم قطعیت و یا تغییر مکان آن در ماشین می‌باشد.

[2] Concrete

[۳] لازم به ذکر است که عملگرهای new و delete برای اینکه در یک هسته قابل استفاده شوند باید سربارگذاری شوند؛ بدین ترتیب قادر خواهیم بود عملگرهایی را بسازیم که از هیپ مخصوص هسته و آنطور که ما انتظار داریم عمل کنند. البته طراحی و صحت سنجی این بخش از هسته ضروری به نظر نمی‌رسد.

```
MACHINE
        Process
SEES
        AddressSpaceType,
        HostMachine,
        ProcessType
VARIABLES
        rt_process,
        rt_process_pid,
        rt_process_parent_pid,
        rt_process_pager_pid,
        rt_process_exman_pid,
        rt_process_address_space,
        rt_process_waiting_ipc_from,
        rt_process_system_status
INVARIANT
        rt_process ∈ ℱ(PROCESS) ∧
        null_process ∉ rt_process ∧
        rt_process_pid ∈ rt_process ⤔ NAT1 ∧
        rt_process_parent_pid ∈ rt_process → NAT1 ∧
        rt_process_pager_pid ∈ rt_process → NAT1 ∧
        rt_process_exman_pid ∈ rt_process → NAT1 ∧
        rt_process_address_space ∈ rt_process → ADDRESS_SPACE ∧
        rt_process_waiting_ipc_from ∈ rt_process → NAT1 ∧
        rt_process_system_status ∈ rt_process → (REGS → NAT) ∧
        ∀pr. (pr ∈ rt_process ⇒ rt_process_address_space (pr) ≠ null_address_space)
INITIALISATION
        rt_process := ∅ ||
        rt_process_pid := ∅ ||
        rt_process_parent_pid := ∅ ||
        rt_process_pager_pid := ∅ ||
        rt_process_exman_pid := ∅ ||
        rt_process_address_space := ∅ ||
        rt_process_waiting_ipc_from := ∅ ||
        rt_process_system_status := ∅
OPERATIONS
        pr ← constructProcessObject (pid, parent_pid, pager_pid, exman_pid, as, from, init) =
        PRE
                pid ∈ NAT1 ∧
                pid ∉ ran (rt_process_pid) ∧
                parent_pid ∈ NAT1 ∧
                pager_pid ∈ NAT1 ∧
                exman_pid ∈ NAT1 ∧
                as ∈ ADDRESS_SPACE ∧
                as ≠ null_address_space ∧
                from ∈ NAT1 ∧
                init : (REGS → NAT)
        THEN
                ANY
                        new_pr
                WHERE
                        new_pr ∈ PROCESS - rt_process ∧
                        new_pr ≠ null_process
                THEN
                        rt_process := rt_process ∪ {new_pr} ||
                        rt_process_pid (new_pr) := pid ||
                        rt_process_parent_pid (new_pr) := parent_pid ||
```

```
                rt_process_pager_pid (new_pr) := pager_pid ||
                rt_process_exman_pid (new_pr) := exman_pid ||
                rt_process_address_space (new_pr) := as ||
                rt_process_waiting_ipc_from (new_pr) := from ||
                rt_process_system_status (new_pr) := init ||
                pr := new_pr
        END
END;

pid ← getPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        pid := rt_process_pid (pr)
END;

parent_pid ← getParentPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        parent_pid := rt_process_parent_pid (pr)
END;

pager_pid ← getPagerPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        pager_pid := rt_process_pager_pid (pr)
END;

exman_pid ← getExmanPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        exman_pid := rt_process_exman_pid (pr)
END;

as ← getAddressSpace (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        as := rt_process_address_space (pr)
END;

wif ← getWaitingIPCFrom (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        wif := rt_process_waiting_ipc_from (pr)
```

END;

ss ← **getSystemStatus** (*pr*) =
PRE
pr ∈ *PROCESS* ∧
pr ∈ *rt_process*
THEN
ss := *rt_process_system_status* (*pr*)
END;

destructProcessObject (*pr*) =
PRE
pr ∈ *PROCESS* ∧
pr ∈ *rt_process*
THEN
rt_process := *rt_process* - {*pr*} ||
rt_process_pid := {*pr*} ⩤ *rt_process_pid* ||
rt_process_parent_pid := {*pr*} ⩤ *rt_process_parent_pid* ||
rt_process_pager_pid := {*pr*} ⩤ *rt_process_pager_pid* ||
rt_process_exman_pid := {*pr*} ⩤ *rt_process_exman_pid* ||
rt_process_address_space := {*pr*} ⩤ *rt_process_address_space* ||
rt_process_waiting_ipc_from := {*pr*} ⩤ *rt_process_waiting_ipc_from* ||
rt_process_system_status := {*pr*} ⩤ *rt_process_system_status*
END
END

پیاده‌سازی ماشین مدیریت اشیاء نوع PROCESS بصورت یک کلاس C++:

```
class Process
{
private:
	int pid;
	int parentPID;
	int pagerPID;
	int exmanPID;
	int waitingIPCFrom;
	SYSTEM_STATE systemState;
	PADDRESS_SPACE address_space;
public:
	Process (const PIDInformation& pids, int waiting,
			const SYSTEM_STATE& initialSystemState, int sz);
	virtual ~Process ();
	PADDRESS_SPACE getAddressSpace () const;
	int getPID () const;
	int getParentPID () const;
	int getPagerPID () const;
	int getExmanPID () const;
	int getWaitingIPCFrom () const;
	const SYSTEM_STATE& getSystemState () const;
	const PADDRESS_SPACE getAddressSpace () const;
};

typedef Process* PPROCESS;
```

در کد ارائه شده در بالا نوع SYSTEM_STATE یک نوع تعریف شده توسط کاربر می‌باشد که در اصل یک آرایه برای نگهداری مقادیر ثبات‌ها می‌باشد؛ در زیر تعریف این نوع را می‌بینیم:

```
typedef unsigned long DWORD;
...
enum {EAX = 0, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, CS, DS, ES, SS,
FS, GS, CR3};
typedef DWORD SYSTEM_STATE [16];
```

از هر یک از اعضای نوع شمارشی می‌توان برای دسترسی عناصر آرایه استفاده کرد. نوع PADDRESS_SPACE اشاره‌گری را به یک شئ از نوع AddressSpace تعریف می‌کند. همانطور که مشاهده می‌کنید بعد از تعریف کلاس Process نیز یک نوع PPROCESS که نوع اشاره‌گر به اشیاء Process است را تعریف کرده‌ایم. در کلاس تعریف شده، سازنده کلاس به هنگام استفاده از عملگر new برای نمونه‌سازی از کلاس فراخوانی شده و شئ‌ایی را از نوع Process مقداردهی اولیه می‌کند، که این دو دقیقاً معادل استفاده از عملگر constructProcessObject در توصیف ماشین Process است. تخریب کننده کلاس هم به هنگام حذف یک شئ ایجاد شده فراخوانی می‌شود که به همراه عملگر delete دقیقاً همان کار عملگر destructProcessObject را انجام می‌دهند. دقت کنید که در مورد ماشین‌های مدیریت اشیاء نمی‌توان از متد B برای پالایش و تولید کد بصورت معمول استفاده کنیم، چرا که این ماشین‌ها در اصل چیزی بیشتر از آن چیزی که واقعاً در یک برنامه شئ‌گرا نوشته می‌شود را دارند که این همان حسابداری اشیاء ایجاد شده و حذف شده است. ما به وسیله جداسازی توصیف‌های مربوط به مدیریت اشیاء و قرار دادن آن‌ها در ماشین‌های جداگانه این امکان را به وجود آورده‌ایم که بتوان در مورد هر نوع از اشیاء صرفاً با استفاده از عملگرهایی ارتباط بر قرار کنیم؛ این ماشین‌ها چیزی جز ایجاد، حذف و دسترسی به خصائص اشیاء چیز دیگری را در خود ندارند که بدین ترتیب خواهیم توانست آن‌ها را بصورت کلاس‌ها C++ پیاده‌سازی کنیم. بدیهی است که در این روند ترجمه به C++ ممکن است خطا و اشتباه وارد شود ولی در این پروژه راه حلی برای این مشکل ارائه نداده‌ایم.

نکته دیگری که باید در مورد پیاده‌سازی سیستم در نظر داشته باشیم این است که باید واسطی را برای برنامه‌های کاربردی در نظر بگیریم. برای حفظ ساختار کلاس‌بندی شده کد، از کلاس‌هایی به عنوان واسط کاربری استفاده خواهیم کرد که دارای اعضای استاتیک هستند، بدین ترتیب قادر خواهیم بود بدون نمونه‌سازی از کلاس‌ها از متدهای آن‌ها استفاده کنیم. این متدها را می‌توان بعداً و در روتین‌های وقفه که به عنوان نکته ورود به فراخوان‌های سیستم عمل می‌کنند، فراخوانی شوند. در ادامه بخشی از کد کلاس SystemInterface را مشاهده می‌کنید که در آن به تعداد فراخوان‌های سیستمی و همنام با

آن‌ها (آن طـوری که در بخش نیازمندی‌های سیستم ذکر شده بود) متد استـاتـیـک وجود دارد که می‌توان از آن‌ها برای انجام وظیفه‌ایی که فراخوان سیستمی متناظر بر عهده دارد استفاده کرد.

```
class SystemInterface
{
private:
      ...
public:
      static inline void createProcess (int pagePID, int exmanPID,
                        const SYSTEM_STATE& initialSystemState,
                        int sz);
      static inline void abortProcess (int pid);
      static inline void schedule ();
      static inline void forceSchedule (int pid);
      static inline void getPID (int& pid, int& parentPID,
                        int& pagerPID, int& exmanPID);
      static inline void mapPage (int pageNo, int pid,
                        int faultyPageNo);
      static inline void grantPage (int pageNo, int pid,
                        int faultyPageNo);
      static inline void reclaimPage (int pageNo);
      static inline void sendMessage (int pid);
      static inline void receiveMessage (int pid);
      static inline void dispatch ();
};
```

همانطور که مشاهده می‌کنید تمامی این متدها بصورت inline تعریف شده‌اند که این باعث می‌شود که بدنه این متدها در هنگام فراخوانی آن‌ها در محل فراخوانی کپی شود، بدین ترتیب با کاهش سربار فراخوانی و قرار دادن پارامترها در پشته به استفاده از این متدها سرعت می‌بخشیم. در هنگام استفاده از فراخوان‌های سیستمی کافی است که در ISR[4] مربوط به وقفه مدیریت کننده فراخوان، هر یک از این متدها را که دلخواه ما است را فراخوانی کنیم.

در ادامه توصیف کامل هر یک از ماشین‌های مدیریت اشیاء پروژه، و بعد از آن پیاده‌سازی ساختار استاتیک پروژه در زبان برنامه نویسی C++ را خواهیم دید.

[4] Interrupt Service Routine

ماشین AddressSpace برای مدیریت اشیاء از نوع AddressSpace:

MACHINE
AddressSpace (*max_pg*, *max_pr*)
CONSTRAINTS
max_pg ∈ **NAT1** ∧
max_pr ∈ **NAT1** ∧
max_pg ≤ 1024
SEES
MapletCodomainType,
AddressSpaceType
VARIABLES
rt_address_space,
rt_address_space_maplets,
rt_address_space_size
INVARIANT
rt_address_space ∈ $\mathcal{F}$(*ADDRESS_SPACE*) ∧
null_address_space ∉ *rt_address_space* ∧
rt_address_space_maplets ∈ *rt_address_space* → **seq** (*MAPLET_CODOMAIN*) ∧
rt_address_space_size ∈ *rt_address_space* → **NAT** ∧
∀*as*. (*as* ∈ *rt_address_space* ⇒ **size** (*rt_address_space_maplets* (*as*)) ≤ *max_pg*) ∧
∀*as*. (*as* ∈ *rt_address_space* ⇒ **size** (*rt_address_space_maplets* (*as*))
= *rt_address_space_size* (*as*))
INITIALISATION
rt_address_space := ∅ ||
rt_address_space_maplets := ∅ ||
rt_address_space_size := ∅
OPERATIONS
as ← **constructAddressSpaceObject** (*maplets*, *sz*) =
PRE
maplets ∈ **seq** (*MAPLET_CODOMAIN*) ∧
null_maplet_codomain ∉ **ran** (*maplets*) ∧
sz ∈ **NAT** ∧
size (*maplets*) = *sz* ∧
size (*maplets*) ≤ *max_pg*
THEN
ANY
new_as
WHERE
new_as ∈ *ADDRESS_SPACE* - *rt_address_space* ∧
new_as ≠ *null_address_space*
THEN
rt_address_space := *rt_address_space* ∪ {*new_as*} ||
rt_address_space_maplets (*new_as*) := *maplets* ||
rt_address_space_size (*new_as*) := *sz* ||
as := *new_as*
END
END;

ml ← **getMapletAt** (*as*, *nn*) =
PRE
as ∈ *ADDRESS_SPACE* ∧
as ∈ *rt_address_space* ∧
nn ∈ **NAT1** ∧
nn ≤ *rt_address_space_size* (*as*)

THEN
ml := *rt_address_space_maplets* (*as*) (*nn*)
END;

destructAddressSpaceObject (*as*) =
PRE
as ∈ *ADDRESS_SPACE* ∧
as ∈ *rt_address_space*
THEN
rt_address_space := *rt_address_space* - {*as*} ||
rt_address_space_maplets := {*as*} ⩤ *rt_address_space_maplets* ||
rt_address_space_size := {*as*} ⩤ *rt_address_space_size*
END
END

ماشین MapletCodomain برای مدیریت اشیاء از نوع MapletCodomain:

MACHINE
MapletCodomain
SEES
MapletCodomainType,
AddressSpaceType,
NatASTupleType,
NatPtrType
VARIABLES
rt_maplet_codomain,
rt_maplet_codomain_real,
rt_maplet_codomain_indirect
INVARIANT
rt_maplet_codomain ∈ $\mathcal{F}$(*MAPLET_CODOMAIN*) ∧
null_maplet_codomain ∉ *rt_maplet_codomain* ∧
rt_maplet_codomain_real ∈ *rt_maplet_codomain* → *NAT_PTR* ∧
rt_maplet_codomain_indirect ∈ *rt_maplet_codomain* → *NAT_AS_TUPLE* ∧
∀*ml*. (*ml* ∈ *rt_maplet_codomain* ⇒ ¬ (*rt_maplet_codomain_real* (*ml*) ≠ *null_nat_ptr* ∧
rt_maplet_codomain_indirect (*ml*) ≠ *null_nat_as_tuple*))
INITIALISATION
rt_maplet_codomain := ∅ ||
rt_maplet_codomain_real := ∅ ||
rt_maplet_codomain_indirect := ∅
OPERATIONS
ml ← **constructMapletCodomainObject** (*real*, *indirect*) =
PRE
real ∈ *NAT_PTR* ∧
indirect ∈ *NAT_AS_TUPLE* ∧
¬ (*real* ≠ *null_nat_ptr* ∧ *indirect* ≠ *null_nat_as_tuple*)
THEN
ANY
new_ml
WHERE
new_ml ∈ *MAPLET_CODOMAIN* - *rt_maplet_codomain* ∧
new_ml ≠ *null_maplet_codomain*
THEN
rt_maplet_codomain := *rt_maplet_codomain* ∪ {*new_ml*} ||
rt_maplet_codomain_real (*new_ml*) := *real* ||
rt_maplet_codomain_indirect (*new_ml*) := *indirect* ||
ml := *new_ml*
END

END;

np ← **getReal** (*ml*) =
PRE
ml ∈ *MAPLET_CODOMAIN* ∧
ml ∈ *rt_maplet_codomain*
THEN
np := *rt_maplet_codomain_real* (*ml*)
END;

setReal (*ml*, *real*) =
PRE
ml ∈ *MAPLET_CODOMAIN* ∧
ml ∈ *rt_maplet_codomain* ∧
real ∈ *NAT_PTR* ∧
¬ (*real* ≠ *null_nat_ptr* ∧ *rt_maplet_codomain_indirect* (*ml*) ≠ *null_nat_as_tuple*)
THEN
rt_maplet_codomain_real (*ml*) := *real*
END;

tp ← **getIndirect** (*ml*) =
PRE
ml ∈ *MAPLET_CODOMAIN* ∧
ml ∈ *rt_maplet_codomain*
THEN
tp := *rt_maplet_codomain_indirect* (*ml*)
END;

setIndirect (*ml*, *indirect*) =
PRE
ml ∈ *MAPLET_CODOMAIN* ∧
ml ∈ *rt_maplet_codomain* ∧
indirect ∈ *NAT_AS_TUPLE* ∧
¬ (*indirect* ≠ *null_nat_as_tuple* ∧ *rt_maplet_codomain_real* (*ml*) ≠ *null_nat_ptr*)
THEN
rt_maplet_codomain_indirect (*ml*) := *indirect*
END;

destructMapletCodomainObject (*ml*) =
PRE
ml ∈ *MAPLET_CODOMAIN* ∧
ml ∈ *rt_maplet_codomain*
THEN
rt_maplet_codomain := *rt_maplet_codomain* - {*ml*} ||
rt_maplet_codomain_real := {*ml*} ⩤ *rt_maplet_codomain_real* ||
rt_maplet_codomain_indirect := {*ml*} ⩤ *rt_maplet_codomain_indirect*
END
END

ماشین NatASTuple برای مدیریت اشیاء از نوع Tuple (Integer, AddressSpace):

MACHINE
NatASTuple
SEES
NatASTupleType,
AddressSpaceType

DEFINITIONS

$INDIRECT\_FRST == (\lambda(pg, as).\ (pg \in \mathbf{NAT} \wedge as \in ADDRESS\_SPACE \mid pg))$;

$INDIRECT\_SCND == (\lambda(pg, as).\ (pg \in \mathbf{NAT} \wedge as \in ADDRESS\_SPACE \mid as))$

VARIABLES

rt_nat_as_tuple,
rt_nat_as_tuple_val

INVARIANT

$rt\_nat\_as\_tuple \in \mathcal{F}(NAT\_AS\_TUPLE) \wedge$
$null\_nat\_as\_tuple \notin rt\_nat\_as\_tuple \wedge$
$rt\_nat\_as\_tuple\_val \in rt\_nat\_as\_tuple \rightarrow (\mathbf{NAT} \times ADDRESS\_SPACE)$

INITIALISATION

$rt\_nat\_as\_tuple := \varnothing \parallel$
$rt\_nat\_as\_tuple\_val := \varnothing$

OPERATIONS

$nt \leftarrow$ **constructNatASTuple** $(ff, ss) =$
PRE
$ff \in \mathbf{NAT} \wedge$
$ss \in ADDRESS\_SPACE \wedge$
$ss \neq null\_address\_space$
THEN
ANY
new_nt
WHERE
$new\_nt \in NAT\_AS\_TUPLE \wedge$
$new\_nt \neq null\_nat\_as\_tuple$
THEN
$rt\_nat\_as\_tuple := rt\_nat\_as\_tuple \cup \{new\_nt\} \parallel$
rt_nat_as_tuple_val $(new\_nt) := (ff \mapsto ss) \parallel$
$nt := new\_nt$
END
END;

$ff \leftarrow$ **getIndFirst** $(nt) =$
PRE
$nt \in NAT\_AS\_TUPLE \wedge$
$nt \in rt\_nat\_as\_tuple$
THEN
$ff := INDIRECT\_FRST\ (rt\_nat\_as\_tuple\_val\ (nt))$
END;

$ss \leftarrow$ **getIndSecond** $(nt) =$
PRE
$nt \in NAT\_AS\_TUPLE \wedge$
$nt \in rt\_nat\_as\_tuple$
THEN
$ss := INDIRECT\_SCND\ (rt\_nat\_as\_tuple\_val\ (nt))$
END;

destructNatASTuple $(nt) =$
PRE
$nt \in NAT\_AS\_TUPLE \wedge$
$nt \in rt\_nat\_as\_tuple$
THEN
$rt\_nat\_as\_tuple := rt\_nat\_as\_tuple - \{nt\} \parallel$
$rt\_nat\_as\_tuple\_val := \{nt\} \lhd rt\_nat\_as\_tuple\_val$
END

END

ماشین NatPtr برای مدیریت اشیاء از نوع Integer (یا بطور معادل نوع * int):

MACHINE
NatPtr
SEES
NatPtrType
VARIABLES
rt_nat_ptr,
rt_nat_ptr_val
INVARIANT
$rt\_nat\_ptr \in \mathcal{F}(NAT\_PTR) \wedge$
$null\_nat\_ptr \notin rt\_nat\_ptr \wedge$
$rt\_nat\_ptr\_val \in rt\_nat\_ptr \rightarrow \mathbf{NAT}$
INITIALISATION
$rt\_nat\_ptr := \varnothing \parallel$
$rt\_nat\_ptr\_val := \varnothing$
OPERATIONS
$np \leftarrow$ **constructNatPtrObject** $(val) =$
PRE
$val \in \mathbf{NAT}$
THEN
ANY
new_np
WHERE
$new\_np \in NAT\_PTR - rt\_nat\_ptr \wedge$
$null\_nat\_ptr \neq new\_np$
THEN
$rt\_nat\_ptr := rt\_nat\_ptr \cup \{new\_np\} \parallel$
rt_nat_ptr_val $(new\_np) := val \parallel$
$np := new\_np$
END
END;

$val \leftarrow$ **getValue** $(np) =$
PRE
$np \in NAT\_PTR \wedge$
$np \in rt\_nat\_ptr$
THEN
$val := rt\_nat\_ptr\_val\ (np)$
END;

destructNatPtrObject $(np) =$
PRE
$np \in NAT\_PTR \wedge$
$np \in rt\_nat\_ptr$
THEN
$rt\_nat\_ptr := rt\_nat\_ptr - \{np\} \parallel$
$rt\_nat\_ptr\_val := \{np\} \lhd rt\_nat\_ptr\_val$
END
END

ماشین Process برای مدیریت اشیاء از نوع Process:

MACHINE
Process

SEES
AddressSpaceType,
HostMachine,
ProcessType
VARIABLES
rt_process,
rt_process_pid,
rt_process_parent_pid,
rt_process_pager_pid,
rt_process_exman_pid,
rt_process_address_space,
rt_process_waiting_ipc_from,
rt_process_system_status
INVARIANT
$rt\_process \in \mathcal{F}(PROCESS) \wedge$
$null\_process \notin rt\_process \wedge$
$rt\_process\_pid \in rt\_process \rightarrowtail \mathbf{NAT1} \wedge$
$rt\_process\_parent\_pid \in rt\_process \rightarrow \mathbf{NAT1} \wedge$
$rt\_process\_pager\_pid \in rt\_process \rightarrow \mathbf{NAT1} \wedge$
$rt\_process\_exman\_pid \in rt\_process \rightarrow \mathbf{NAT1} \wedge$
$rt\_process\_address\_space \in rt\_process \rightarrow ADDRESS\_SPACE \wedge$
$rt\_process\_waiting\_ipc\_from \in rt\_process \rightarrow \mathbf{NAT1} \wedge$
$rt\_process\_system\_status \in rt\_process \rightarrow (REGS \rightarrow \mathbf{NAT}) \wedge$
$\forall pr.\ (pr \in rt\_process \Rightarrow rt\_process\_address\_space\ (pr) \neq null\_address\_space)$
INITIALISATION
$rt\_process := \varnothing \parallel$
$rt\_process\_pid := \varnothing \parallel$
$rt\_process\_parent\_pid := \varnothing \parallel$
$rt\_process\_pager\_pid := \varnothing \parallel$
$rt\_process\_exman\_pid := \varnothing \parallel$
$rt\_process\_address\_space := \varnothing \parallel$
$rt\_process\_waiting\_ipc\_from := \varnothing \parallel$
$rt\_process\_system\_status := \varnothing$
OPERATIONS
$pr \leftarrow$ **constructProcessObject** $(pid, parent\_pid, pager\_pid, exman\_pid, as, from, init) =$
PRE
$pid \in \mathbf{NAT1} \wedge$
$pid \notin \mathbf{ran}\ (rt\_process\_pid) \wedge$
$parent\_pid \in \mathbf{NAT1} \wedge$
$pager\_pid \in \mathbf{NAT1} \wedge$
$exman\_pid \in \mathbf{NAT1} \wedge$
$as \in ADDRESS\_SPACE \wedge$
$as \neq null\_address\_space \wedge$
$from \in \mathbf{NAT1} \wedge$
$init : (REGS \rightarrow \mathbf{NAT})$
THEN
ANY
new_pr
WHERE
$new\_pr \in PROCESS - rt\_process \wedge$
$new\_pr \neq null\_process$
THEN
$rt\_process := rt\_process \cup \{new\_pr\} \parallel$
rt_process_pid $(new\_pr) := pid \parallel$
rt_process_parent_pid $(new\_pr) := parent\_pid \parallel$
rt_process_pager_pid $(new\_pr) := pager\_pid \parallel$
rt_process_exman_pid $(new\_pr) := exman\_pid \parallel$

```
                    rt_process_address_space (new_pr) := as ||
                    rt_process_waiting_ipc_from (new_pr) := from ||
                    rt_process_system_status (new_pr) := init ||
                    pr := new_pr
        END
END;

pid ← getPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        pid := rt_process_pid (pr)
END;

parent_pid ← getParentPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        parent_pid := rt_process_parent_pid (pr)
END;

pager_pid ← getPagerPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        pager_pid := rt_process_pager_pid (pr)
END;

exman_pid ← getExmanPID (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        exman_pid := rt_process_exman_pid (pr)
END;

as ← getAddressSpace (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        as := rt_process_address_space (pr)
END;

wif ← getWaitingIPCFrom (pr) =
PRE
        pr ∈ PROCESS ∧
        pr ∈ rt_process
THEN
        wif := rt_process_waiting_ipc_from (pr)
END;
```

ss ← **getSystemStatus** (*pr*) =
PRE
pr ∈ *PROCESS* ∧
pr ∈ *rt_process*
THEN
ss := *rt_process_system_status* (*pr*)
END;

destructProcessObject (*pr*) =
PRE
pr ∈ *PROCESS* ∧
pr ∈ *rt_process*
THEN
rt_process := *rt_process* - {*pr*} ||
rt_process_pid := {*pr*} ⩤ *rt_process_pid* ||
rt_process_parent_pid := {*pr*} ⩤ *rt_process_parent_pid* ||
rt_process_pager_pid := {*pr*} ⩤ *rt_process_pager_pid* ||
rt_process_exman_pid := {*pr*} ⩤ *rt_process_exman_pid* ||
rt_process_address_space := {*pr*} ⩤ *rt_process_address_space* ||
rt_process_waiting_ipc_from := {*pr*} ⩤ *rt_process_waiting_ipc_from* ||
rt_process_system_status := {*pr*} ⩤ *rt_process_system_status*
END
END

ماشین Queue برای مدیریت اشیاء از نوع Queue:

MACHINE
Queue (*max_queue_size*)
CONSTRAINTS
max_queue_size ∈ **NAT** ∧
max_queue_size ≤ 1024
SEES
QueueType,
ProcessType
VARIABLES
rt_queue,
rt_queue_processes,
rt_queue_size
INVARIANT
rt_queue ∈ $\mathcal{F}$(*QUEUE*) ∧
null_queue ∉ *rt_queue* ∧
rt_queue_processes ∈ *rt_queue* → **iseq** (*PROCESS*) ∧
rt_queue_size ∈ *rt_queue* → **NAT** ∧
∀*qq*. (*qq* ∈ *rt_queue* ⇒ *rt_queue_size* (*qq*) ≤ *max_queue_size*)
INITIALISATION
rt_queue := ∅ ||
rt_queue_processes := ∅ ||
rt_queue_size := ∅
OPERATIONS
qq ← **constructQueueObject** =
BEGIN
ANY
new_q
WHERE
new_q ∈ *QUEUE* - *rt_queue* ∧
new_q ≠ *null_queue*
THEN

```
                rt_queue := rt_queue ∪ {new_q} ||
                rt_queue_processes (new_q) := ∅ ||
                rt_queue_size (new_q) := 0 ||
                qq := new_q
        END
END;

enQueue (qq, pp) =
PRE
        qq ∈ QUEUE ∧
        qq ∈ rt_queue ∧
        pp ∈ PROCESS ∧
        pp ∉ ran (rt_queue_processes (qq)) ∧
        rt_queue_size (qq) < max_queue_size
THEN
        rt_queue_processes (qq) := rt_queue_processes (qq) ← pp ||
        rt_queue_size (qq) := rt_queue_size (qq) + 1
END;

pp ← deQueue (qq) =
PRE
        qq ∈ QUEUE ∧
        qq ∈ rt_queue ∧
        rt_queue_size (qq) > 0
THEN
        pp := first (rt_queue_processes (qq)) ||
        rt_queue_processes (qq) := tail (rt_queue_processes (qq)) ||
        rt_queue_size (qq) := rt_queue_size (qq) - 1
END;

nn ← getSize (qq) =
PRE
        qq ∈ QUEUE ∧
        qq ∈ rt_queue
THEN
        nn := rt_queue_size (qq)
END;

pp ← getAt (qq, nn) =
PRE
        qq ∈ QUEUE ∧
        qq ∈ rt_queue ∧
        nn ∈ NAT1 ∧
        rt_queue_size (qq) ≥ nn
THEN
        pp := last (rt_queue_processes (qq) ↑ nn)
END;

pp ← remAt (qq, nn) =
PRE
        qq ∈ QUEUE ∧
        qq ∈ rt_queue ∧
        nn ∈ NAT1 ∧
        rt_queue_size (qq) ≥ nn
THEN
```

```
            pp := last (rt_queue_processes (qq) ↑ nn) ||
            rt_queue_size (qq) := rt_queue_size (qq) - 1 ||
            rt_queue_processes (qq) := front (rt_queue_processes (qq) ↑ nn) ^
                                                    (rt_queue_processes (qq) ↓ nn)
        END;

        destructQueueObject (qq) =
        PRE
            qq ∈ QUEUE ∧
            qq ∈ rt_queue
        THEN
            rt_queue := rt_queue - {qq} ||
            rt_queue_processes := {qq} ⩤ rt_queue_processes ||
            rt_queue_size := {qq} ⩤ rt_queue_size
        END
END
```

در ادامه پیاده‌سازی ساختار استاتیک سیستم را خواهیم دید که به زبان C++ نوشته شده است:

```
/*****************************************************************************
*****************************************************************************
                                    Sequence.h
                                  by Ali Ghanbari-2012
*****************************************************************************
*****************************************************************************/

#pragma once

template <class T, int _BASE_>
class Sequence;

template <class T, int _BASE_>
class SequenceNode
{
        friend class Sequence <T, _BASE_>;
private:
        mutable T data;
        SequenceNode <T, _BASE_>* next;
public:
        SequenceNode () {}
        SequenceNode (const T& data, SequenceNode <T, _BASE_>* next)
                                                {this->data = data, this->next = next;}
};

template <class T, int _BASE_>
class Sequence
{
private:
        SequenceNode <T, _BASE_>* head;
        int _size;
        mutable T dummy;
        SequenceNode <T, _BASE_>* getItem (int index) const;
public:
        Sequence ();
        Sequence (const Sequence <T, _BASE_>& r);
        virtual ~Sequence ();
        int size () const {return _size;}
        int remAt (int index);
        T& operator[] (int index) const;
        const Sequence <T, _BASE_>& operator= (const Sequence <T, _BASE_>& r);
        void* append (const T& data);
        void* append (const Sequence <T, _BASE_>& seq);
        void* prepend (const T& data);
        void* prepend (const Sequence <T, _BASE_>& seq);
        T& at (int index) const {return (*this) [index];}
        T& first () const {return (*this) [0 + _BASE_];}
        T& last () const {return head ? head->data : dummy;}
};

template <class T, int _BASE_>
Sequence <T, _BASE_>::Sequence () {
        head = 0;
        _size = 0;
}

template <class T, int _BASE_>
Sequence <T, _BASE_>::Sequence (const Sequence <T, _BASE_>& r) {
        head = 0;
        _size = 0;
        append (r);
}

template <class T, int _BASE_>
void* Sequence <T, _BASE_>::append (const T& data) {
        head = new SequenceNode <T, _BASE_> (data, head);
        _size ++;
        return head;
}

template <class T, int _BASE_>
SequenceNode <T, _BASE_>* Sequence <T, _BASE_>::getItem (int index) const {
        SequenceNode <T, _BASE_>* p = head;
```

```
        if (index >= _size || index < 0) return 0;
        for (int i = 1; i < (_size - index); i ++) p = p->next;
        return p;
}

template <class T, int _BASE_>
int Sequence <T, _BASE_>::remAt (int index) {
        SequenceNode <T, _BASE_>* p;
        SequenceNode <T, _BASE_>* prev;

        index -= _BASE_;
        p = getItem (index);
        if (!p) return 0;
        prev = getItem (index + 1);
        if (prev) prev->next = p->next;
        else head = p->next;
        delete p;
        _size --;
        return 1;
}


template <class T, int _BASE_>
T& Sequence <T, _BASE_>::operator[] (int index) const {
        SequenceNode <T, _BASE_>* p = getItem (index - _BASE_);

        if (!p) return dummy;
        return p->data;
}

template <class T, int _BASE_>
void* Sequence <T, _BASE_>::prepend (const T& data) {
        SequenceNode <T, _BASE_>* p = getItem (0);

        if (!p) return 0;
        p->next = new SequenceNode <T, _BASE_> (data, 0);
        if (p->next) {
                _size ++;
                return p->next;
        }
        return 0;
}

template <class T, int _BASE_>
void* Sequence <T, _BASE_>::append (const Sequence <T, _BASE_>& seq) {
        void *res = 0;

        for (int i = 0; i < seq._size; i ++) {
                res = append (seq [i + _BASE_]);
                if (!res) return 0;
        }
        return res;
}

template <class T, int _BASE_>
void* Sequence <T, _BASE_>::prepend (const Sequence <T, _BASE_>& seq) {
        void *res = 0;

        for (int i = seq._size - 1; i >= 0; i --) {
                res = prepend (seq [i + _BASE_]);
                if (!res) return 0;
        }
        return res;
}

template <class T, int _BASE_>
const Sequence <T, _BASE_>& Sequence <T, _BASE_>::operator=
                                                        (const Sequence <T, _BASE_>& r) {
        SequenceNode <T, _BASE_>* p = head;

        while (p) {
                p = p->next;
                delete head;
                head = p;
        }
        head = 0;
```

```
        _size = 0;
        append (r);
}

template <class T, int _BASE_>
Sequence <T, _BASE_>::~Sequence () {
        SequenceNode <T, _BASE_>* p = head;

        while (p) {
                p = p->next;
                delete head;
                head = p;
        }
}

/*********************************************************************************************
*********************************************************************************************
                                        Queue.h
                                    by Ali Ghanbari-2012
*********************************************************************************************
*********************************************************************************************/

#pragma once

#include "BasicDefs.h"
#include "Sequence.h"
#include "Process.h"

class Queue
{
private:
        Sequence <PPROCESS, 1> processes;
public:
        Queue ();
        virtual ~Queue ();
        PPROCESS enQueue (PPROCESS pr);
        PPROCESS deQueue ();
        PPROCESS remAt (int index);
        const PPROCESS getAt (int index) const;
        int size () const;
};

typedef Queue* PQUEUE;

/*********************************************************************************************
*********************************************************************************************
                                        Queue.cpp
                                    by Ali Ghanbari-2012
*********************************************************************************************
*********************************************************************************************/

#include "Queue.h"

Queue::Queue () {
        processes [1] = 0;
}

PPROCESS Queue::enQueue (PPROCESS pr) {
        processes.append (pr);
}

PPROCESS Queue::deQueue () {
        PPROCESS pr = processes.at (1);

        processes.remAt (1);
        return pr;
}

const PPROCESS Queue::getAt (int index) const {
        return processes [index];
}

int Queue::size () const {
        return processes.size ();
}

Queue::~Queue () {
```

```
        for (int i = 1; i <= processes.size (); i ++)
                delete processes.at (i);
}

/*********************************************************************************
*********************************************************************************
                                        BasicDefs.h
                                    by Ali Ghanbari-2012
*********************************************************************************
*********************************************************************************/

#pragma once

#define NULL    ((void*)0)
#define MAX_PR  100
#define MAX_PG  100

typedef unsigned long DWORD;
typedef unsigned short WORD;
typedef unsigned char BYTE;
enum {EAX = 0, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, CS, DS, ES, SS, FS, GS, CR3};
typedef DWORD SYSTEM_STATE [16];

/*********************************************************************************
*********************************************************************************
                                        Process.h
                                    by Ali Ghanbari-2012
*********************************************************************************
*********************************************************************************/

#pragma once

#include "AddressSpace.h"
#include "BasicDefs.h"
#include "Message.h"

class Process
{
private:
        int pid;
        int parentPID;
        int pagerPID;
        int exmanPID;
        int waitingIPCFrom;
        SYSTEM_STATE systemState;
        PADDRESS_SPACE address_space;
public:
        Process (int pid, int parentPID, int pagerPID, int exmanPID, int waiting,
                                        const SYSTEM_STATE& initialSystemState,
                                        int sz);
        virtual ~Process ();
        int getPID () const;
        int getParentPID () const;
        int getPagerPID () const;
        int getExmanPID () const;
        int getWaitingIPCFrom () const;
        const SYSTEM_STATE& getSystemState () const;
        const PADDRESS_SPACE getAddressSpace () const;
};

typedef Process* PPROCESS;

/*********************************************************************************
*********************************************************************************
                                        Process.cpp
                                    by Ali Ghanbari-2012
*********************************************************************************
*********************************************************************************/

#include "Process.h"

Process::Process (int pid, int parentPID, int pagerPID, int exmanPID, int waiting,
                                        const SYSTEM_STATE& initialSystemState,
                                        int sz) {
        address_space = new AddressSpace (sz);
        this->pid = pid;
        this->parentPID = parentPID;
```

```
        this->pagerPID = pagerPID;
        this->exmanPID = exmanPID;
        this->waitingIPCFrom = waiting;
        this->systemState = initialSystemState;
}

int Process::getPID () const {
        return pid;
}

int Process::getParentPID () const {
        return parentPID;
}

int Process::getPagerPID () const {
        return pagerPID;
}

int Process::getExmanPID () const {
        return exmanPID;
}

int Process::getWaitingIPCFrom () const {
        return waitingIPCFrom;
}

const SYSTEM_STATE& Process::getSystemState () const {
        return systemState;
}

const PADDRESS_SPACE Process::getAddressSpace () const {
        return address_space;
}

virtual Process::~Process () {
        delete address_space;
}

/*****************************************************************************
******************************************************************************
                                    AddressSpace.h
                                by Ali Ghanbari-2012
******************************************************************************
*****************************************************************************/

#pragma once

#include "MapletCodomain.h"
#include "BasicDefs.h"
#include "Sequence.h"

class AddressSpace
{
private:
  Sequence <PMAPLET_CODOMAIN, 0> maplets;
public:
  AddressSpace (int sz);
  virtual ~AddressSpace ();
  Sequence <PMAPLET_CODOMAIN, 0>& getMaplets ();
};

typedef AddressSpace* PADDRESS_SPACE;

/*****************************************************************************
******************************************************************************
                                    AddressSpace.h
                                by Ali Ghanbari-2012
******************************************************************************
*****************************************************************************/

#pragma once

#include "MapletCodomain.h"
#include "BasicDefs.h"
#include "Sequence.h"

class AddressSpace
```

```
{
private:
        Sequence <PMAPLET_CODOMAIN, 0> maplets;
public:
        AddressSpace (int sz);
        virtual ~AddressSpace ();
        Sequence <PMAPLET_CODOMAIN, 0>& getMaplets ();
};

typedef AddressSpace* PADDRESS_SPACE;

/*****************************************************************************
******************************************************************************
                                    AddressSpace.cpp
                                  by Ali Ghanbari-2012
******************************************************************************
*****************************************************************************/

#include "AddressSpace.h"

AddressSpace::AddressSpace (int sz) {
        for (int i = 0; i < sz; i ++)
                maplets.append (new MapletCodomain (NULL, NULL));
}

Sequence <PMAPLET_CODOMAIN>&  AddressSpace::getMaplets () {return maplets;}

AddressSpace::~AddressSpace () {
        for (int i = 0; i < maplets.size (); i ++)
                delete maplets.at (i);
}

/*****************************************************************************
******************************************************************************
                                    MapletCodomain.h
                                  by Ali Ghanbari-2012
******************************************************************************
*****************************************************************************/

#pragma once

#include "AddressSpace.h"

typedef struct
{
        int index;
        PADDRESS_SPACE address_space;
} INDIRECT_TUPLE;

class MapletCodomain
{
private:
        int* real;
        INDIRECT_TUPLE* indirect;
public:
        MapletCodomain (int* real, INDIRECT_TUPLE* indirect);
        const int* getReal () const {return real;}
        void setReal (const int* real);
        INDIRECT_TUPLE* getIndirect () const {return indirect;}
        void setIndirect (const INDIRECT_TUPLE* indirect);
        virtual ~MapletCodomain ();
};

typedef MapletCodomain* PMAPLET_CODOMAIN;

/*****************************************************************************
******************************************************************************
                                    MapletCodomain.cpp
                                  by Ali Ghanbari-2012
******************************************************************************
*****************************************************************************/

#include "MapletCodomain.h"

MapletCodomain::MapletCodomain (int* real, INDIRECT_TUPLE* indirect) {
        this->real = real;
        this->indirect = indirect;
```

```
}

void MapletCodomain::setReal (const int* real) {
        if (this->real) delete (this->real);
        this->real = real;
}

void MapletCodomain::setIndirect (const INDIRECT_TUPLE* indirect) {
        if (this->indirect) delete (this->indirect);
        this->indirect = indirect;
}

MapletCodomain::~MapletCodomain () {
        if (real) delete real;
        if (indirect) delete indirect;
}
```

۷-۲- نتیجه گیری

در این فصل مشاهده کردیم که چگونه می‌توان اطلاعات اضافی در توصیف‌ها که صرفاً به خاطر سادگی و آزادی در توصیف در نظر گرفته شده بودند را حذف کرد؛ و نیز عدم قطعیت که باز یکی دیگر از ابزارهای ساده سازی توصیف می‌باشد را طی مراحل پالایش حذف یا تغییر مکان داد. همانطور که در ابتدای این فصل نیز به آن اشاره شد، پیاده‌سازی کامل تمام بخش‌های سیستم زمانی بیش از یک سال و تیمی بزرگتر از یک نفر را می‌طلبد، اما این بدین معنا نیست که ما وظیفه خود را در جهت پیاده‌سازی سیستم انجام نداده‌ایم؛ همانطور که مشاهده نمودید ما در این پروژه راهکار پیاده‌سازی را ارائه دادیم و صرفاً با صرف وقت بیشتر می‌توان پیاده‌سازی را انجام داد. در انتهای این فصل نیز روشی را برای مدیریت در زمان اجرای اشیاء کلاس‌های مختلف ارائه دادیم، و نیز تعدادی از کلاس‌ها را زبان C++ پیاده‌سازی کردیم.