第 5 回の課題について

外部断片化に対する一つの解法としてメモリの詰直し（コンパクション）があり、これは使用中の領域を移動させて一つの大きな空き領域を作るものです。教科書 p. 129 にあるように、

「しかし, 割当て領域のアドレスを要求プログラムに返している場合はこの方式はとれない.」

としてこれが実現困難である主な理由は以下になります。

「割当て領域のアドレスを要求プログラムに返している」というのは、プログラム実行中に malloc のような処理によってメモリ領域 M を割り当て、そこのアドレス Ma をプログラムに返すことであり、プログラム内では Ma によって M を直接参照・操作します。ここで、Ma を格納した領域（ポインタ）については、アドレスを格納しているということは実行時に OS からは分からないので、このプロセスの領域を移動すると、（動的再配置がなければ）たとえば 1000 番地にあるデータ x がポインタ p によって参照されている場合、詰直しによって x が 550 番地に移動されたときに、元々は 1000 というアドレス値を保持していた p の内容を 550 に変更することができず、内容が誤ってしまうことになります。

【提出いただいた解答へのコメント】

- 割当て領域のアドレスを保持するプログラムでは詰直しを行うと正しいデータが参照できなくなる
- 詰直しされてしまうとデータが元のアドレスから移動され、本来返されるアドレスと異なってしまう

- プログラムとのズレが生じてしまう / アドレスの整合性がとれなくなる / 無効になる
- アドレスが変わってしまうので、中身が変わってしまう
  返すアドレスが固定である

などといった解答を多くいただきました。「割当て領域のアドレスを要求プログラムに返している場合はこの方式はとれない. 」の意味として妥当ですが、「正しいデータが参照できない」や「ズレ」、「整合性がとれない」ことの内容について、ポインタのように具体例をあげた説明があると分かりやすいのではと思います。

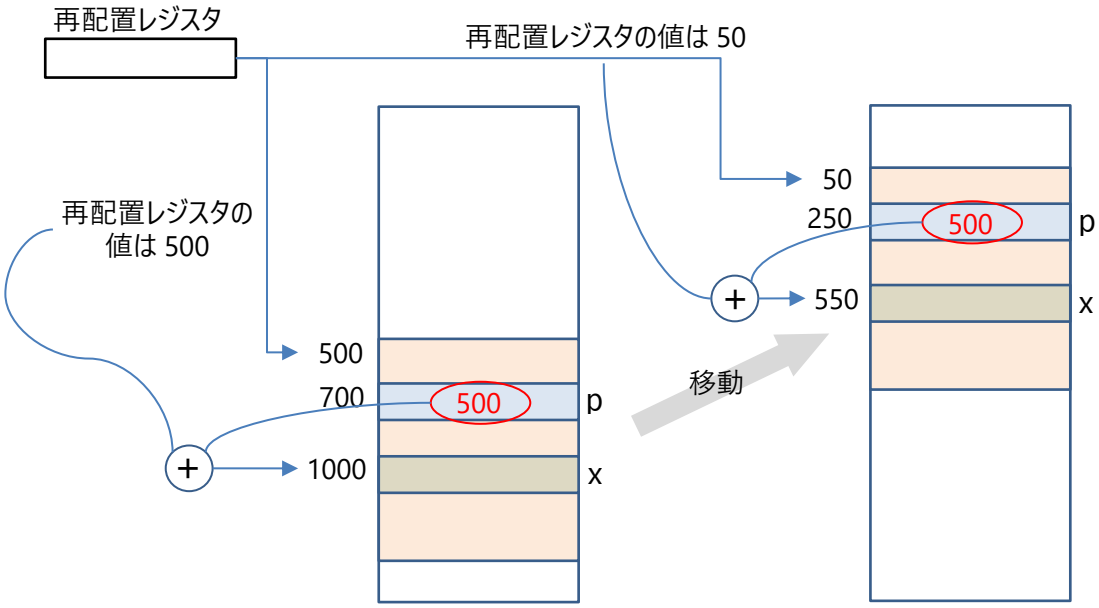
また、普通のデータでは再配置ローダ（静的再配置）で大丈夫であるのに対して、この場合はうまくいかないことに関する記述があると、さらによいのではと思います。

講義でははっきりと説明していませんでしたが、これを解決するには動的な再配置が必要になります。静的な再配置はロード時に再配置ローダで行われるものですが、動的再配置はプログラムの実行時に行われ、再配置レジスタ（リロケーションレジスタ）と呼ばれるハードウェア機構を用いて、メモリのアクセスすべてを再配置レジスタからのオフセットによって行うものです。

静的な再配置では、上記のポインタ p で指されている 1000 番地のデータが 550 番地に移動さ

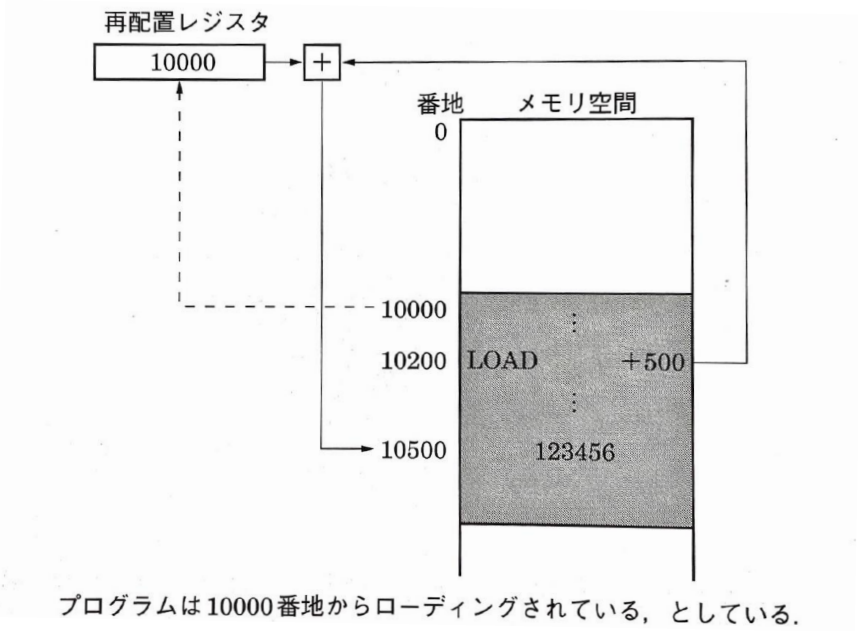
れる問題には対応できませんが、動的な再配置があれば大丈夫です。

二段階で参照するという解決策を示していただいた解答もありました。



上図で p の内容として入る 500 は、このプログラムが 0 番地から始まるものとして得た x のアドレスで、プログラム（プロセスの領域）の先頭（論理的に 0 番地）からのオフセット（すなわち、実際のメモリアドレスでなく論理的なアドレス）であると考えられます。再配置レジスタにプログラムの開始位置を保持し、常にその値にオフセットを加えたもので実際のアドレスを得る機構を用いると、 プログラムが移動されても正しいアドレスでアクセスできます。

教科書 p.122「3. 再配置レジスタによる方法」も見ておいてください。



> コンパイラまたはリンカが作成するオブジェクトプログラムは，あたかも 0 番地からローディングされるようなプログラムにしておく． プログラムの実行時，プログラムに割り当てられたメモリ領域の先頭アドレスを再配置レジスタにセットする．これはオペレーティングシステムが行うか，またはコンピュータによってはプログラムが自分で行う． プログラム実行時の実際のアドレスは，プログラム中のアドレスに再配置レジスタの内容を加えたもの（その操作はハードウェアにより自動的に行われる）となる．

このようにアドレスをどこかの起点(この場合は再配置レジスタ)からのオフセットで表現することが、今回、第 6 回で学ぶページングでは、ページの先頭からのオフセットで表現することに対応することになります。すなわち、ページングは動的な再配置を行っている(ページテーブルのエントリが再配置レジスタというイメージになります)ことになり、実メモリ上で「連続したアドレス」という制約がなくなるので、外部断片化が発生しません。